

## THESIS / THÈSE

### MASTER EN SCIENCES INFORMATIQUES

#### Le ROS dans l'architecture de courrier électronique X400

Laurent, Vincent

*Award date:*  
1987

*Awarding institution:*  
Université de Namur

[Link to publication](#)

#### General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

#### Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

18/12/87  
18/12/87

**Le ROS dans l'architecture de  
courrier électronique X400**

**Vincent Laurent**

**Septembre 1987**

Nous remercions la société SIEMENS  
SOFTWARE de nous avoir accueillis pour  
notre stage dans ses laboratoires à  
Munich et nous exprimons notre gratitude  
à notre Directeur, Monsieur Ph. Van  
Bastelaer ainsi qu'à toutes les  
personnes nous ayant aidés à la  
réalisation de ce mémoire.



## Table des matières

---

|                                                                                           |    |
|-------------------------------------------------------------------------------------------|----|
| INTRODUCTION                                                                              | 1  |
| CHAPITRE 1 Les réseaux informatiques                                                      |    |
| 1.0. Introduction                                                                         | 4  |
| 1.1. La classification des réseaux                                                        | 6  |
| 1.1.1. Les réseaux de la recherche                                                        | 7  |
| 1.1.2. Les réseaux de firmes                                                              | 8  |
| 1.1.3. Les réseaux de coopératives                                                        | 9  |
| 1.1.4. Les réseaux commerciaux                                                            | 10 |
| 1.1.5. Les méta-réseaux                                                                   | 10 |
| 1.2. Le réseau Arpanet                                                                    | 11 |
| 1.3. Le réseau EARN                                                                       | 12 |
| CHAPITRE 2 Généralités sur le modèle MHS                                                  |    |
| 2.0. Introduction                                                                         | 16 |
| 2.1. Le modèle MHS                                                                        | 17 |
| 2.1.1. La description générale du modèle                                                  | 17 |
| 2.1.2. Les perspectives physiques                                                         | 22 |
| 2.2. Quelques remarques sur le principe<br>de dénomination, d'adressage,<br>et de routage | 25 |
| 2.2.1. Nom primitif-nom descriptif                                                        | 26 |
| 2.2.2. Le nom E/D                                                                         | 27 |
| 2.2.3. L'adresse E/D                                                                      | 27 |
| 2.2.4. Le principe de routage                                                             | 28 |
| 2.3. La représentation en couches<br>du modèle MHS                                        | 29 |



|                                                           |    |
|-----------------------------------------------------------|----|
| 2.3.1. La description en couches                          | 31 |
| CHAPITRE 3 L'entité SDE et son protocole de communication |    |
| 3.1. L'entité SDE                                         | 37 |
| 3.1.1. Le RDS (Remote Operation Server)                   | 38 |
| 3.1.2. L'AM (Association Manager)                         | 39 |
| 3.1.3. Le RTS (Reliable Transfer Server)                  | 39 |
| 3.2. Le protocole de soumission et livraison, P3          | 40 |
| 3.2.1. Les opérations de gestion                          | 43 |
| 3.2.1.1. L'opération Register                             | 43 |
| 3.2.1.2. L'opération Control                              | 44 |
| 3.2.1.3. L'opération Change-Password                      | 44 |
| 3.2.2. Les opérations de soumission                       | 45 |
| 3.2.2.1. L'opération Submit                               | 45 |
| 3.2.2.2. L'opération Probe                                | 45 |
| 3.2.2.3. L'opération Cancel                               | 46 |
| 3.2.3. L'opération Deliver                                | 46 |
| 3.2.4. L'opération Notify                                 | 47 |
| CHAPITRE 4 Le mécanisme de Remote Operation               |    |
| 4.0. Introduction                                         | 48 |
| 4.1. Le déroulement d'une opération à distance            | 49 |
| 4.2. L'aspect fonctionnel du RDS                          | 50 |
| 4.3. L'environnement RDS dans le modèle OSI               | 51 |
| 4.4. Les classes d'utilisation du RDS                     | 52 |
| 4.4.1. Les classes d'association-RDS                      | 53 |
| 4.4.2. La classification des opérations à                 | 54 |

distance

|                                                                     |    |
|---------------------------------------------------------------------|----|
| 4.4.2.1. Le type de l'opération                                     | 55 |
| 4.4.2.2. Le mode de l'opération                                     | 56 |
| 4.4.3. Les classes de l'opération à distance                        | 57 |
| 4.4.4. Les combinaisons entre classe<br>d'association-RDS et classe | 58 |

d'opérations

|                                                             |    |
|-------------------------------------------------------------|----|
| 4.5. Les concepts de Remote Operation et de<br>Remote Error | 59 |
| 4.5.1. Le concept de Remote Operation                       | 60 |
| 4.5.2. Le concept de Remote Error                           | 61 |
| 4.6. Les types d'OPDU                                       | 62 |
| 4.6.1. L'OPDU de demande                                    | 62 |
| 4.6.2. L'OPDU de résultat                                   | 63 |
| 4.6.3. L'OPDU d'erreur                                      | 64 |
| 4.6.4. L'OPDU de rejet                                      | 65 |

CHAPITRE 5 Le système de courrier électronique  
proposé

par SIEMENS

|                                               |    |
|-----------------------------------------------|----|
| 5.0. Introduction                             | 68 |
| 5.1. L'architecture générale du EM de SIEMENS | 68 |
| 5.2. Les caractéristiques du module TPDU      | 71 |
| 5.2.1. La fonction du module TPDU             | 72 |
| 5.2.2. L'environnement du TPDU                | 73 |
| 5.2.3. La justification du TPDU               | 74 |

CHAPITRE 6 Le travail réalisé chez SIEMENS

|                                          |    |
|------------------------------------------|----|
| 6.1. La présentation générale du travail | 75 |
|------------------------------------------|----|



|                                                        |    |
|--------------------------------------------------------|----|
| 6.1.1. Le but du travail                               | 75 |
| 6.1.2. La justification du travail                     | 76 |
| 6.1.3. Les étapes du travail                           | 77 |
| 6.2. L'analyse du TPDU                                 | 78 |
| 6.2.1. La définition des opérations                    | 78 |
| 6.2.2. La structure de liste                           | 79 |
| 6.3. La spécification externe des fonctions<br>du TPDU | 81 |
| 6.3.1. La fonction tp2_001_open                        | 81 |
| 6.3.2. La fonction tp2_002_read                        | 82 |
| 6.3.3. La fonction tp2_003_write                       | 83 |
| 6.3.4. La fonction tp2_004_position                    | 84 |
| 6.3.5. La fonction tp2_005_close                       | 85 |
| 6.3.6. La fonction tp2_006_delete                      | 86 |
| 6.4. La spécification interne des fonctions<br>du TPDU | 87 |
| 6.4.1. La fonction tp2_001_open                        | 89 |
| 6.4.2. La fonction tp2_002_read                        | 89 |
| 6.4.3. La fonction tp2_003_write                       | 90 |
| 6.4.4. La fonction tp2_004_position                    | 90 |
| 6.4.5. La fonction tp2_005_close                       | 90 |
| 6.4.6. La fonction tp2_006_delete                      | 91 |
| 6.5. Le test du nouveau TPDU                           | 92 |
| 6.6. La conclusion du travail                          | 93 |
| CONCLUSION GENERALE                                    | 95 |
| BIBLIOGRAPHIE                                          | 97 |



## INTRODUCTION

-----

Au cours de cette dernière décennie, la télécommunication nationale et internationale est devenue de plus en plus importante. Ce développement a engendré de nouveaux services dans le domaine informatique.

Le courrier électronique, encore appelé messagerie électronique, fait partie de cette nouvelle génération des services. Il permet à ses utilisateurs de s'échanger des messages à travers le monde. Il constitue donc la poste du futur.

Cependant, étant donné l'échelle internationale de cette messagerie, il était opportun qu'un organisme consultatif propose un standard aux constructeurs de logiciels. Les différents produits réalisés en suivant ce modèle pourraient ainsi s'interconnecter plus facilement et sans adjoindre d'onéreux systèmes pour les adapter les uns aux autres.

En 1984, le Comité Consultatif International pour les Télégraphes et la Téléphonie ( CCITT ) publia, X400/84, une série de recommandations dans lesquelles le modèle Message Handling System ( MHS ) est défini comme le standard pour le courrier électronique.

Pour compléter notre formation en informatique, nous avons trouvé enrichissant de nous spécifier dans cette branche des télécommunications.



Préalablement à la conception de ce document, nous avons été accueilli par SIEMENS SOFTWARE à Munich pour effectuer un stage de fin d'études.

Pendant cette période, un travail relatif à l'application courrier électronique a été réalisé.

Le domaine de réflexion abordé dans ce mémoire sera donc cette branche de la télécommunication internationale et nous tenterons d'expliquer, de façon plus précise, certaines techniques rencontrées lors de la réalisation du travail.

Dans un premier chapitre, nous mentionnerons quelques réseaux informatiques notoires; le réseau américain Arpanet ainsi que le réseau européen EARN seront présentés.

L'application courrier électronique est modélisée par le modèle MHS exposé dans la norme X400/84; le chapitre 2 présentera ce modèle.

Le travail effectué se situe, plus particulièrement, au niveau de l'entité appelée Submission and Delivery Entity ( SDE ) dans le modèle MHS; au cours du chapitre 3, nous détaillerons ses différents composants et nous spécifierons le protocole de soumission et livraison, P3, qui régit la communication du SDE avec l'entité Message Transfer Agent Entity ( MTAE ) .

Le chapitre 4 analysera la technique du Remote Operation qui est utilisée dans le modèle MHS pour structurer le protocole d'application interactif P3. Après avoir décrit la notation et défini les concepts

employés, nous spécifierons les primitives de service utilisées pour le transfert fiable des unités de données.

La suite du document aura comme cadre de référence l'architecture de courrier électronique proposée par SIEMENS.

Le chapitre 5 nous présentera cette architecture.

Le chapitre 6 spécifiera le module Transfer Protocol Data Unit ( TPDU ) qui fut le siège du travail, il expliquera l'objectif de celui-ci et en détaillera les diverses étapes.

Au terme de ce mémoire, se dégagera une réflexion basée sur le mécanisme d'opérations à distance. Nous inviterons le lecteur à poursuivre des recherches sur l'application de la technique du Remote Operation pour structurer d'autres protocoles.



### 1.0. Introduction

---

Le recours à l'ordinateur pour traiter automatiquement de l'information est de plus en plus souvent employé. De grandes banques de données se sont développées un peu partout à travers le monde. Les utilisateurs de ces systèmes éprouvaient un besoin grandissant de communiquer entre eux. Grâce au progrès effectué dans le domaine de la micro-électronique, les connexions entre ces systèmes ont pu être établies. C'est ainsi que fut créé le concept de réseau informatique.

Nous pouvons définir un réseau informatique comme un ensemble d'ordinateurs connectés par des moyens de transmission et qui utilisent des protocoles communs pour communiquer entre eux.

La figure 1.2 nous présente un croquis d'un réseau informatique.

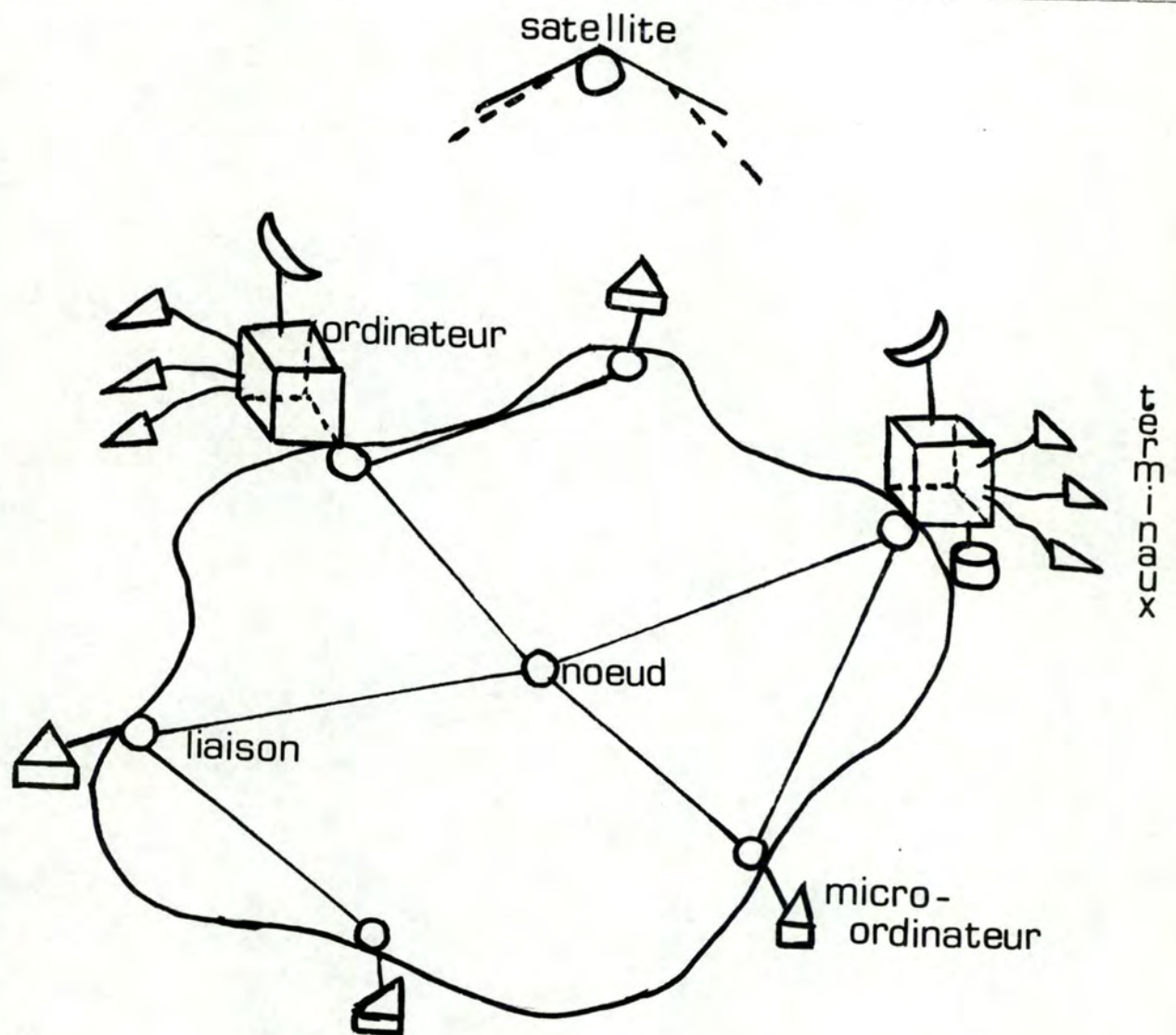


fig 1.2: Le croquis d'un réseau informatique

Ces réseaux peuvent avoir une étendue allant du site local jusqu'à un niveau international. Les moyens de transmission utilisés pour connecter les ordinateurs variant du micro jusqu'au super ordinateur, sont des fils téléphoniques, des cables coaxiaux, des fibres optiques, des satellites. Sur ces lignes de transmission, des protocoles sont utilisés: ils se différencient en fonction de leur vitesse, de leur fiabilité de



transmission et de façon plus générale en fonction de leurs fonctionnalités. Les services rendus par ces réseaux sont nombreux: nous citerons le système de courrier électronique, le système de fichiers distribués ou encore d'exécution de travaux à distance.

Tous ces facteurs de distinction permettent d'établir de multiples classifications.

Pour notre part, nous reprendrons les réseaux qui favorisent le développement de la recherche dans ce domaine informatique et pour cette raison, nous les qualifierons de notoires. La classification adoptée sera celle reprise par Quarterman et Hoskins. ( Quart 86 ). Elle nous paraît, en effet, fortuite car elle fait ressortir l'omniprésence du service de courrier électronique sur les classes de réseaux évoquées. Ce service de communication de messages se révèle ainsi comme essentiel pour les utilisateurs connectés aux différents réseaux.

La section 1 de ce chapitre présentera les catégories de réseaux issues de cette classification. La section 2 sera consacrée à Arpanet, un réseau créé par l'Advanced Research Projects Agency ( ARPA ), un service du département américain de la défense. La section 3 nous fera connaître EARN, un réseau européen d'ordinateurs universitaires.

#### 1.1. La classification des réseaux

-----

Dans cette section, nous allons traiter de la



classification des réseaux informatiques. Elle est basée sur des considérations administratives: un réseau appartient à une catégorie selon les intentions poursuivies par les personnes qui en sont propriétaires, selon l'origine des fonds qui le financent, selon les raisons qui ont poussé les utilisateurs à se connecter au réseau.

En prenant cette base de discrimination, nous faisons apparaître cinq catégories de réseaux informatiques:

- les réseaux de la recherche
- les réseaux de firmes
- les réseaux de coopératives
- les réseaux commerciaux
- les méta-réseaux.

#### 1.1.1. Les réseaux de la recherche

-----

La plupart des premiers réseaux qui sont apparus, appartiennent à la catégorie des réseaux de la recherche. Ils sont le fruit d'importants projets poursuivis dans le domaine de la télé-informatique. Cependant, beaucoup de ces réseaux servent encore maintenant de support au développement de la recherche dans de nombreux domaines, Arpanet en est l'exemple le plus connu.

De tels réseaux sont souvent gérés par les autorités publiques et financés par des subventions publiques. Leurs utilisateurs sont essentiellement des chercheurs participant aux projets.



Outre les principaux services qu'un réseau de transmission de données peut offrir ( interrogation à distance de bases de données, système de fichiers répartis,... ), les réseaux de la recherche possèdent également une messagerie électronique. Celle-ci permet aux chercheurs de s'échanger des messages et de pouvoir collaborer étroitement à la réalisation de projets.

Nous noterons que le coût à payer pour bénéficier des services de ces réseaux est très important.

Les réseaux CSNET, MFENET en sont d'autres exemples repris dans cette catégorie.

#### 1.1.2. Les réseaux de firmes

-----

Avant de développer le paragraphe consacré aux réseaux de firmes, il convient de rappeler que notre classification reprend uniquement les réseaux notoires, c'est-à-dire ceux qui sont un terrain fertile pour le développement de la recherche. Le lecteur ne s'étonnera pas de l'absence de réseaux importants comme le réseau de la Générale de Banque, le réseau de grandes entreprises industrielles ... pour lesquels l'activité poursuivie est essentiellement à caractère économique.

Cependant de grands constructeurs de produits informatiques comme IBM, XEROX, DEC, AT&T ont mis en oeuvre leur propre réseau. Certains de ces réseaux sont locaux, limités à quelques immeubles, cependant d'autres ont une échelle internationale. La gestion ainsi que le financement de ces réseaux reviennent aux firmes qui en



sont propriétaires. Les utilisateurs sont, pour la plupart, des employés travaillant dans ces firmes.

Parmi ces réseaux, nous citerons Ethernet inventé par XEROX, Easynet de DEC fournissant les services tels que le transfert de fichiers entre systèmes, l'accès de fichiers à distance, le courrier électronique, le partage de ressources entre systèmes, l'exécution de procédures à distance,....

La firme IBM possède son propre réseau appelé VNET. Il est composé de deux réseaux: RSCS ( Remote Spooling and Communication Subsystem ) et PUM ( Passthru UM ).

#### 1.1.3. Les réseaux de coopératives

-----

Certains réseaux ont été fondés par des utilisateurs poursuivant des travaux avec des intérêts similaires. Les principaux réseaux sont BITNET aux Etats-Unis, NET NORTH au Canada, EARN et EUNET en Europe et Asia-Net au Japon.

Ces réseaux établis dans l'environnement universitaire, permettent aux utilisateurs d'échanger des messages entre deux appareils connectés. Ces réseaux sont le plus souvent un mélange de matériels venant de divers constructeurs ainsi que de différents systèmes d'exploitation ( MS-DOS, UNIX,... ).

La gestion de ces réseaux est généralement décentralisée et est aux mains des différents utilisateurs s'y étant connectés. Les redevances ne sont pas collectées par une organisation centrale mais sont plutôt payées par chaque groupe d'utilisateurs aux autres



groupes auxquels il s'est connecté.

Cependant certains réseaux reçoivent d'importants subsides des firmes les équipant en matériel. Ainsi IBM aide financièrement BITNET et sponsorise totalement EARN.

#### 1.1.4. Les réseaux commerciaux

-----

Dans la classe de réseaux commerciaux, nous regrouperons les différents réseaux publics nationaux. De tels réseaux sont orientés vers un grand public, et dans un but lucratif, ils offrent à ces utilisateurs des applications télématiques telles que Télétex, Vidéotex, Télémail, Télex,....

Ces téléservices sont supportés par les différents réseaux téléphoniques nationaux et aussi par les réseaux de commutation par paquets comme DCS en Belgique, TRANSPAC en France, DATEX-P en Allemagne fédérale,... ( Mac 83. p.545 à 588 )

Leur gestion est toujours centralisée. Leur financement provient de redevances payées par les utilisateurs pour le raccordement et des coûts d'exploitation variant en fonction du temps d'utilisation du réseau ou du volume de l'information transmise sur celui-ci.

#### 1.1.5. Les méta-réseaux

-----

Plusieurs projets visent à étendre les groupes

d'utilisateurs des réseaux existants pour former un système de communication d'une échelle supérieure. Ce méta-réseau engloberait plusieurs réseaux déjà en exploitation et fournirait ainsi une communication entre utilisateurs connectés sur différents réseaux.

Un seul projet a abouti et a donné le méta-réseau Cenet. Ce méta-réseau a pour but de favoriser la recherche et le développement en informatique, en permettant une meilleure collaboration intercontinentale entre les chercheurs travaillant dans ce domaine. Le service qu'il offre est principalement un échange de messages par un courrier électronique.

Dans la suite du chapitre, nous présenterons brièvement deux réseaux déjà évoqués: Arpanet et EARN.

#### 1.2. Le réseau Arpanet

---

Le réseau Arpanet fut créé en 1968 par la branche informatique du département américain de la défense, Arpanet est le plus ancien réseau hétérogène de grande envergure ( Quart 86 et Mah 84 ). Son objectif est de réaliser des communications entre des centres de recherche et des universités en leur permettant l'accès partagé à des ressources onéreuses telles que des puissants CPUs, des périphériques avec facilités graphiques,....

Arpanet, entièrement financé par l'ARPA, est un réseau de commutation par paquets. Il comporte actuellement cinquante sept noeuds de commutation appelés IMPs. Il



possède également des liaisons permanentes. Il fut réalisé par la société Bolt Beranek and Newman Inc ( BBN ) à Boston.

Les cinq cents ordinateurs faisant partie de ce réseau sont reliés par des connexions à 64 Kb/s à travers les USA, et par des connexions satellites vers l'Europe ( Stuttgart et Londres notamment ). Des passerelles permettent à Arpanet de communiquer avec de nombreux autres réseaux ( Satnet, BBN-Net,... ).

Un service de transfert de messages est offert par le réseau de transmission de données. Pour gérer la communication entre processus, un composant de transport, Network Control Program ( NCP ), est situé dans chaque ordinateur raccordé. NCP utilise un protocole de transport de bout en bout. ( Mac 83 p.266, 267, 577 ).

La gestion automatique de l'acheminement du message évite à l'utilisateur de connaître la topologie du réseau. En effet, l'expéditeur d'un message ne doit pas avoir à l'esprit la liste des noeuds par lesquels son message va transiter avant d'arriver chez son destinataire.

Un inconvénient est le coût de connexion à ce réseau; il s'élève à environ cent vingt mille dollars par an.

### 1.3. Le réseau EARN

-----

EARN ( European Academic Research Network ) est un réseau européen d'ordinateurs universitaires calqué sur Bitnet. Il fournit une liaison par lignes louées entre



cent cinquante ordinateurs répartis dans cent institutions européennes. Parmi les pays connectés, nous citerons l'Espagne, l'Italie, la France, la Suisse, l'Allemagne fédérale, l'Angleterre, la Suède, l'Irlande, ... ( Mah 84 p 449 ).

La gestion et les services techniques sont traités par un ordinateur central pour chaque pays connecté. IBM fournit la connexion à Bitnet et paye le coût des livraisons.

Dans chaque état connecté à EARN, il est possible de développer un sous-réseau; ainsi les allemands ont développé un sous-réseau d'une vingtaine de sites. D'autre part, les universitaires concernés ont commencé à développer des liaisons avec Cenet.

EARN fournit un service de transfert de messages et de fichiers. Il permet également la soumission de travaux à distance. Un service de courrier électronique ( INFO ) a également été installé.

Cependant EARN souffre de l'hostilité du comité européen des PTT souhaitant qu'IBM utilise des liaisons X25 et non pas des lignes louées.

A travers cette classification des réseaux, il ressort que la présence d'un système de courrier électronique est constante dans la gamme des services offerts par tous ces réseaux.

Avant d'exposer le modèle MHS, proposé par le CCITT, pour la mise en oeuvre d'un système de courrier électronique, nous terminerons ce chapitre en décrivant



les principaux services offerts par ce système.

La fonction principale d'un système de courrier électronique consiste à accepter d'un terminal un ensemble de données alphanumériques ( appelé un message ) présentées selon un format déterminé et de remettre ce message au destinataire identifié par une adresse. Les utilisateurs d'un système peuvent ainsi communiquer entre eux en s'échangeant des messages.

Ce système fournit également un certain nombre de services complémentaires tels que:

- la remise différée dans le cas d'un utilisateur occupé ou absent. Les messages en attente sont mémorisés sur un fichier " boîte aux lettres ", stockés sur le site du destinataire. Ce dernier peut les visualiser automatiquement dès la mise en service de son terminal ou lorsqu'il le désire au cours d'une session de travail. Ce service est particulièrement intéressant pour les réseaux internationaux par le fait du décalage horaire.

- la diffusion, c'est-à-dire, la remise d'un même message à un ensemble de destinataires identifiés par une liste d'adresses

- la définition de différents niveaux de priorité pour un message ( urgent, normal, différé ). Ces niveaux correspondent à des différentes valeurs du délai d'acheminement du message.

- l'offre généralisée, c'est-à-dire, la mise à la disposition d'un ensemble de messages aux usagers du système avec éventuellement certaines restrictions

d'accès

- le retour d'un accusé de remise du message au destinataire. Il informe l'émetteur que son message a bien été remis au destinataire.

- la conversion de codes ou de format du message lorsque les terminaux émetteurs et destinataires ne sont pas compatibles.

Les options offertes par une messagerie électronique étant rappelées, il convient maintenant de décrire un modèle proposé aux constructeurs de logiciels pour la réalisation d'un système de courrier électronique.



### 2.0. Introduction

---

Dans le but de pouvoir facilement interconnecter des matériels hétérogènes émanant de constructeurs concurrents, il était impératif qu'un organisme consultatif développe un modèle qui puisse servir de canevas à la définition de protocoles standards.

En 1979, le CCITT reconnaît comme un standard la fameuse architecture en sept couches proposée par l'International Standards Organisation ( ISO ). En 1984, le CCITT publie la série de recommandations X400 proposant le Message Handling System ( MHS ), comme modèle pour la réalisation d'un système de messagerie électronique. Ce modèle s'insère au sommet de l'architecture ISO.

Ces standards sont implantés dans les produits fournis par les principaux constructeurs informatiques européens tels que Bull, Thomson, Olivetti, Philips, Siemens,....

Ce chapitre est une introduction au MHS. Il comprend une description de ce modèle, de ces éléments de service et quelques notions sur le principe de dénomination et d'adressage. Il introduit également les protocoles définis dans X400 et l'architecture en couches relative au modèle MHS.

## 2.1. Le modèle MHS

---

Le système de traitement de messages défini dans la recommandation X400, fournit un service de courrier électronique " Store and Forward ". ( Mye 84, Cun 83, Mal 86 et Mau 86 ).

Ce concept signifie que le mécanisme d'acheminement des messages est indirect: l'itinéraire de ceux-ci est décomposé en étapes entrecoupées de noeuds au sein desquels les messages sont temporairement stockés. Le modèle MHS offre à ses utilisateurs la possibilité d'envoyer et de recevoir des messages. Il est constitué de différents composants fonctionnels travaillant ensemble. L'organisation de ceux-ci est structurée en couches selon la technique du modèle de référence Open System Interconnection ( OSI ).

### 2.1.1. La description générale du modèle

---

Une perspective fonctionnelle du modèle MHS est décrite à la figure 2.a . L'utilisateur tantôt appelé " expéditeur " s'il envoie un message, tantôt " destinataire " s'il en reçoit un, est soit une personne soit un processus sur un système informatique.

Les messages transmis par le MHS sont essentiellement des messages interpersonnels. Cependant, il serait possible d'imaginer le traitement d'autres types de



messages tels que des messages bancaires, ... .

Ainsi, lorsque l'utilisateur désire envoyer un message à un ou plusieurs destinataire(s), il le prépare avec l'assistance de l'UA auquel il est connecté.

Un Uber Agent ( UA ) est un ensemble de processus ( un éditeur, un système de gestion de fichiers, ... ) qui sont utilisés pour créer les messages ou pour gérer leur stockage.

Pendant la préparation du message, l'expéditeur de celui-ci communique avec son UA au moyen d'un équipement d'entrée/sortie ( un clavier, une machine facsimile, ... ) et lui transmet le message ainsi que les paramètres concernant son transfert ( liste des adresses des destinataires, priorité du message, ... ).

Selon l'équipement d'entrée/sortie utilisé, le cadre dans lequel le message est encodé peut être différent.

L'UA expéditeur, ayant construit le message, le soumet au Message Transfer System ( MTS ) afin que celui-ci le délivre aux UAs destinataires concernés par ce message.

Le MTS se compose d'un certain nombre de Message Transfer Agents ( MTAs ). Ils travaillent ensemble et fournissent les éléments de service de transfert de messages dont le principal est l'acheminement du message jusqu'aux UAs destinataires désirés. Ces derniers mettent en forme le message avant de le présenter aux utilisateurs destinataires qui peuvent en prendre connaissance moyennant un instrument de sortie ( écran, imprimante, ... ).

L'ensemble des UAs et des MTAs constituent le MHS.

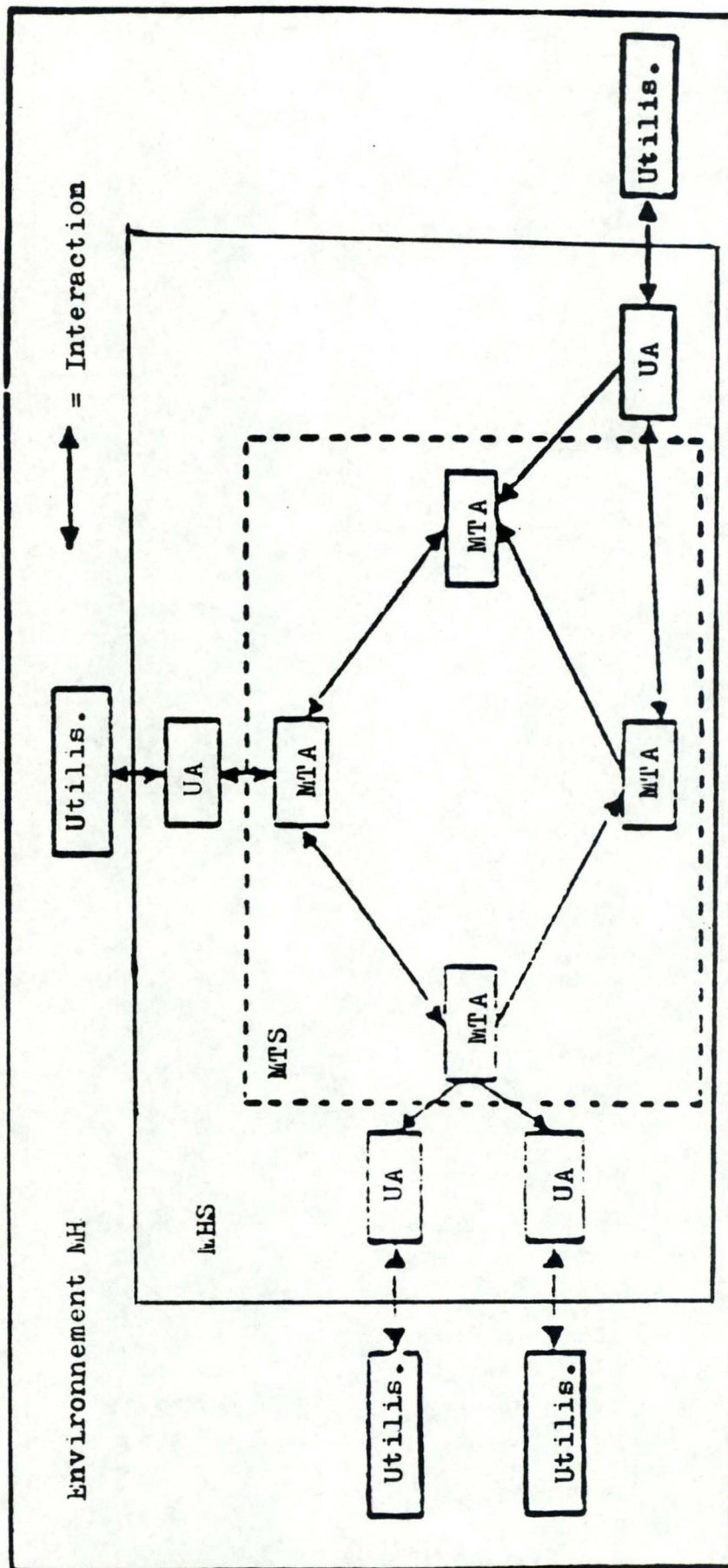


fig 2.2: La représentation fonctionnelle du modèle MHS



Le Message Handling Environment regroupe le MHS avec tous les utilisateurs de celui-ci.

La figure 2.b montre la structure élémentaire d'un message tel que traité par le MTS.

Il se compose de deux parties:

- L'enveloppe est constituée de l'information nécessaire au transfert du message; le MTA la construit en se servant des paramètres passés par l'expéditeur. La valeur de ceux-ci renferme la liste d'adresses des destinataires, l'adresse de l'expéditeur, la priorité du message ,... .

- Le contenu qui englobe d'une part l'entete ou nous trouvons de l'information de controle ajoutant une signification supplémentaire au message ( par exemple: le sujet du message, la référence à une correspondance antérieure,... ) et d'autre part le corps ou nous trouvons l'information devant être communiquée par l'expéditeur à un ou plusieurs destinataire(s). Ce contenu est fabriqué par l'UA en collaboration avec l'utilisateur.

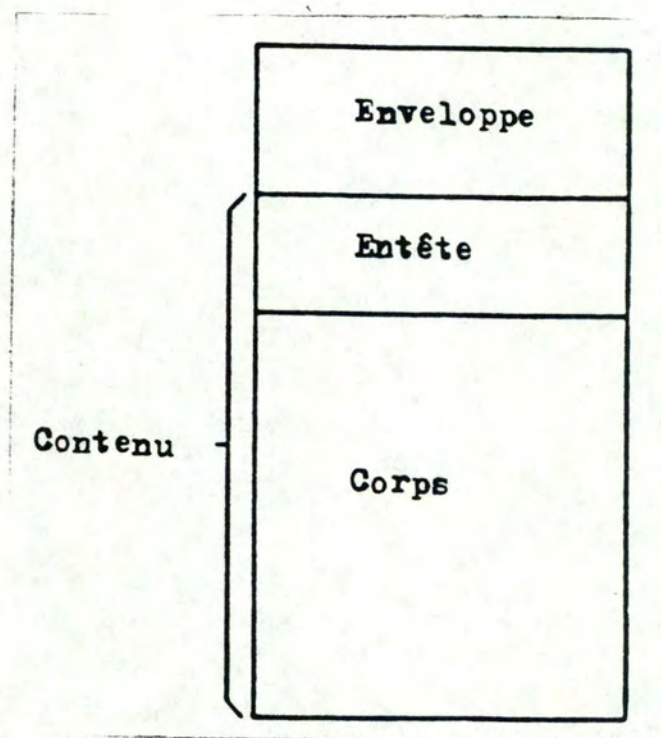


fig 2.b : La structure de base du message

Lors du transfert d'un message de l'expéditeur vers un destinataire, apparaissent deux interactions UA-MTS:

- La soumission par laquelle l'UA expéditeur soumet un message avec ses paramètres au MTS

- La livraison par laquelle le MTS livre le message à l'UA destinataire.

Pendant ces deux interactions, la responsabilité du message est transférée de l'UA au MTS et réciproquement.



Les MTAs transfèrent les messages contenant tout type d'information codée en binaire et n'interprètent ni n'altèrent le contenu de ceux-ci sauf si l'expéditeur l'exige au moyen d'un élément de service particulier ( par exemple: la conversion du contenu d'un message d'un code ASCII vers un code TTX ). Les MTAs sont donc théoriquement transparents au contenu du message.

#### 2.1.2. Les perspectives physiques

-----

Comme nous l'avons défini au paragraphe précédent un UA est un ensemble de processus d'application dans un système informatique. Il contient au minimum les fonctions nécessaires pour interagir avec le MTS au moyen des procédures de soumission et de livraison. Il appartient à une classe définie suivant le type du contenu du message qu'il peut traiter.

Plusieurs configurations d'implantation d'un UA et d'un MTA sont possibles comme le montre la figure 2.c.

Un UA et un MTA peuvent être implantés dans un même système. Nous avons alors un système groupé dans lequel l'UA accède aux éléments de service du MT en interagissant directement avec le MTA se trouvant dans le même système.

D'une façon alternative, un UA peut aussi être implanté dans un système physiquement séparé. Nous sommes alors en présence d'un UA autonome qui, pour communiquer avec le MTS, doit accéder à un MTA implanté dans un

système éloigné. Cette communication s'effectue au moyen d'un protocole standardisé décrit dans la norme X410 ( X410 ). Ce protocole qui dans le cadre du MHS est appelé " Protocole P3 " règle au bit près la communication entre l'UA éloigné et le MTA.

Cette architecture sera évoquée de façon plus détaillée dans la suite de ce rapport car elle est le coeur de ce travail.

Enfin, il est aussi possible pour un MTA d'être implanté dans un système sans UA. Nous le nommerons alors MTA autonome.



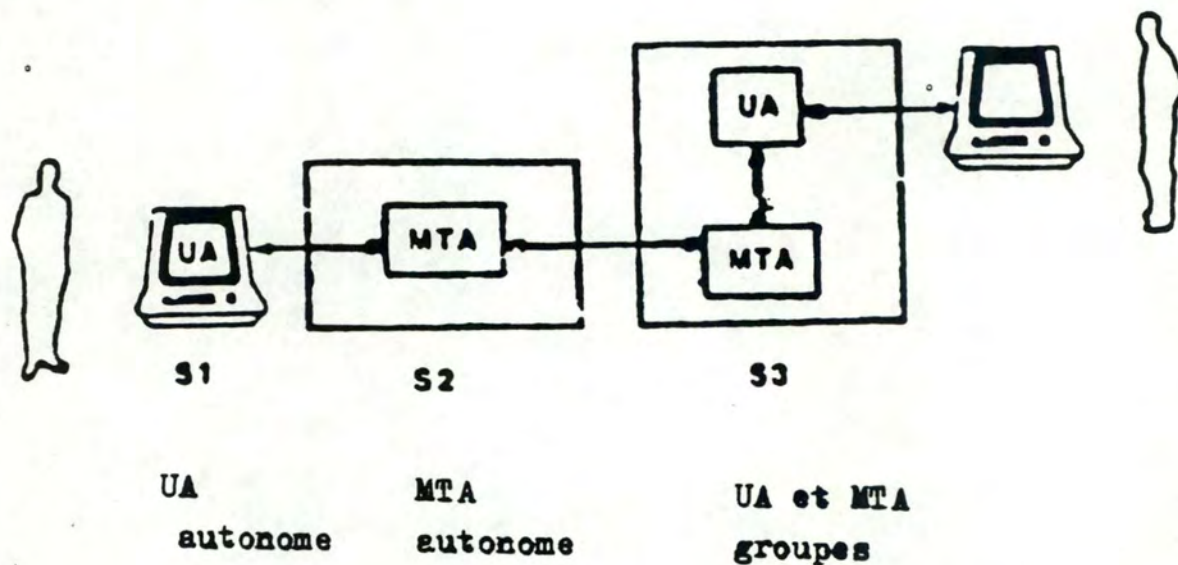


fig 2.c: Les diverses perspectives physiques du modèle MHS

Ces systèmes physiques différents peuvent être connectés par des liaisons dédiées ou par tout type de réseau.

## 2.2. Quelques remarques sur le principe de

-----  
dénomination, d'adressage et de routage  
-----

-Le MHS peut être considéré comme un système réparti: en effet, il fournit à ses utilisateurs, situés sur des sites géographiquement distants, la possibilité de s'échanger des messages.

Le caractère réparti entraîne certaines contraintes lors de l'attribution du nom et de l'adresse pour chaque utilisateur du système. Le principe de dénomination est celui selon lequel chaque utilisateur se voit affecter une liste d'attributs comme son nom, sa profession, sa qualification au sein de l'institution dans laquelle il se trouve. D'autre part pour qu'un message soit transféré de son expéditeur vers le destinataire désiré, il doit être acheminé selon un mécanisme de routage déterminant le chemin que le message doit suivre.

Dans ce paragraphe, nous expliquerons le concept de nom E/D, d'adresse E/D. L'abréviation E/D signifie Expéditeur/Destinataire et rappelle que chaque utilisateur du système est tantôt expéditeur, tantôt destinataire d'un message. La suite de ce paragraphe évoquera également le principe de routage d'un message.



### 2.2.1. Nom primitif-nom descriptif

-----

Nous appelons " Nom primitif " un nom assigné par une autorité qui attribue les noms: pour chaque autorité considérée séparément, elle ne distribue jamais deux fois le même nom à l'intérieur de son propre environnement.

Par exemple: A l'intérieur d'un département d'une entreprise, le nom des employés est un nom primitif restreint au contexte du département auquel il appartient.

Nous appelons " Nom descriptif " un nom qui identifie un et un seul utilisateur dans le monde MHS. La propriété d'identifiant est globale et indépendante du contexte propre de l'autorité nommante. Un nom descriptif identifie de façon univoque un utilisateur en spécifiant un ou plusieurs de ses attributs: ceux-ci sont des caractéristiques de l'utilisateur telles que la firme et le département dans lequel il travaille, la fonction occupée, ou encore l'adresse de l'immeuble où siège cette firme,....

Par exemple:- Firme: construction logiciel

Namur

- Département: ventes

- Fonction: président.

Dans un monde MHS où il n'y a qu'un seul utilisateur correspondant à cette liste; ce nom est un nom descriptif pour celui-ci.

### 2.2.2. Le nom E/D

-----

L'entité principale requérant un nom est l'utilisateur du MHS qui envoie et reçoit des messages. Cependant celui-ci se trouvant en dehors du contexte MHS, il est inconnu du système ( cfr figure 2.a du modèle fonctionnel MHS ). C'est pourquoi le nom descriptif est attribué à l'UA de l'utilisateur. La liste des attributs ainsi associée à chaque UA est appelée le nom E/D de l'UA. Pour envoyer un message à un correspondant, l'utilisateur expéditeur doit connaître le nom E/D ( O/R name ) de l'UA du destinataire.

### 2.2.3. L'adresse E/D

-----

L'adresse E/D de l'UA se compose des attributs spécifiant la localisation géographique de l'UA et des valeurs de ceux-ci. Ces attributs sont le numéro de la rue, le nom de la rue, la région, le pays, ... . Ainsi l'UA est identifié par l'adresse E/D lorsque cette entité est spécifiée par sa localisation géographique. Les valeurs de l'adresse E/D de l'UA correspondent à l'adresse du point de rattachement de celui-ci au MHS.

L'adresse E/D de l'UA est un nom descriptif pour cet UA car à cette adresse ne correspond qu'un seul UA dans



le monde MHS.

Entre nom E/D et adresse E/D, la relation est la suivante: chaque adresse E/D est un nom E/D. En effet, le nom E/D identifie une entité en spécifiant des caractéristiques générales de celle-ci telles que sa fonction au sein d'une organisation, son nom, son adresse, sa profession, ... . Par contre l'adresse E/D ne reprend que les caractéristiques géographiques de l'entité, c'est-à-dire son adresse. Donc, les adresses E/D forment un sous-ensemble des noms E/D.

#### 2.2.4. Le principe de routage

-----

L'itinéraire d'un message est l'information qui décrit le chemin qui doit être suivi par le message lorsqu'il progresse à travers le MTS pour aller de l'UA expéditeur jusqu'à l'UA destinataire. Il est établi sur base de l'adresse E/D du destinataire. Celle-ci doit donc être correctement transmise et spécialement en ce qui concerne l'ensemble des valeurs la constituant. Le message est routé de MTA en MTA tout au long de l'itinéraire indiqué dans son adresse E/D. Il est important de noter que chacun de ces MTAs détermine le MTA suivant auquel le message doit être transféré, mais ne prend aucune autre décision au-delà de cette détermination. Donc aucun MTA ne tente d'établir le chemin complet. Chaque MTA a une responsabilité temporaire du message. Elle se limite entre le moment où il reçoit le message et le moment où il le transfère vers le MTA suivant appartenant à

l'itinéraire du message.

Par exemple: Si une adresse E/D est formée  
comme suit:

Numéro = 11

Nom-rue = Reagan-President

Nom-ville = Los Angeles

Nom-région = Californie

Nom-pays = Etats-Unis

Un message destiné à cet utilisateur sera  
transmis du MTA expéditeur vers un MTA aux  
Etats-Unis, arrivé dans ce pays il sera envoyé  
vers un MTA dans la région de la Californie  
puis vers un MTA dans la ville Los Angeles,...

Cet exemple montre la construction  
incrémentale de l'itinéraire du message de MTA  
en MTA à travers le MTS jusqu'à arriver au MTA  
destinataire.

### 2.3. La représentation en couches du modèle MHS

-----

Avant de décrire la représentation du modèle MHS, nous  
signalons que celle-ci est profondément remise en  
question dans la recommandation X400/88 qui sera publiée  
l'an prochain.

Par contre en analysant la recommandation actuelle,  
nous constatons que la structuration en couches est  
encore fort dominante.



Cette représentation du MHS reprend les différentes fonctions mises en évidence précédemment dans le modèle fonctionnel: soumission et livraison du message d'une part, routage de celui-ci d'autre part.

Cette représentation permet d'appliquer au MHS les techniques et les principes repris dans le modèle de référence Open System Interconnection ( OSI ) ( Brun 86 ). Elle rend aussi possible l'identification des différentes entités, des protocoles et des interfaces de service nécessaires. Les protocoles MHS identifiés peuvent être rattachés à des sous-couches spécifiques se localisant au niveau application dans le modèle OSI. Le principe général de la modélisation en couches repose sur certaines règles qui aident à l'identification des couches. Nous les évoquons brièvement ci-après:

- Les couches sont définies de façon à regrouper au maximum les fonctions similaires
- Les couches sont définies en vue de minimiser les interactions entre les couches
- Les couches sont définies de telle sorte que l'utilisation de différents protocoles au sein d'une même couche est permise et qu'elle n'affecte pas la définition du service de celle-ci
- Les couches sont définies de façon optimale: en effet nous déterminons avec ces règles un nombre minimum de couches
- Les couches sont définies pour que chacune d'elles ajoute un nouveau service à ceux déjà fournis par les couches inférieures



- Les couches sont définies de sorte que chacune d'elles requière un ou plusieurs protocole(s). Chacun d'entre eux standardise la communication entre deux entités au sein d'une même couche.

Conscients de ces quelques considérations, nous pouvons analyser la découpe en couches du modèle MHS.

### 2.3.1. La description en couches

-----

La structure en couches du MHS s'insère au sommet du modèle de référence OSI. Celui-ci est réparti en sept couches qui sont d'un point de vue fonctionnel regroupées en trois niveaux.

Les trois premières couches décrivent un standard pour le service réseau vu comme le moyen de communication mutuelle entre des systèmes ouverts.

Les trois couches suivantes sont relatives à la septième. Elles décrivent une communication " end-to-end " entre les systèmes ouverts. La couche quatre ( transport ) fournit un complément fonctionnel au service réseau selon l'exigence de la couche application ( couche sept ) afin d'obtenir un transport fiable des données. La couche cinq ( session ) procure les services permettant aux processus appelants et appelés de dialoguer. Son objectif est, d'abord, de négocier les accords du dialogue préalablement aux interactions et, ensuite, sur base des décisions prises, de réaliser la synchronisation des opérations effectuées. La couche six ( présentation )



fournit un service permettant, après négociation entre les correspondants portant sur leur possibilité de formatage de l'information, de transmettre sous forme standard, des données de façon différentes par des processus d'application. Dans le cas du MHS le service de cette couche est réduit à rien; en effet, le format des messages à envoyer à travers le système est spécifié dans les moindres détails par la recommandation X409.

La couche sept ( application ) est la couche la plus externe du modèle de référence; elle permet la compréhension et l'exécution de commandes liées aux processus d'application. Dans le contexte de ce rapport, nous nous intéressons à l'application de la messagerie électronique modélisée par le MHS.

Si nous examinons la couche sept, proposée par le MHS, pour cette application, nous remarquons différents types d'entités ainsi que la présence de protocoles permettant de régler la communication entre celles-ci.

Utilisant les six couches inférieures, le MHS peut établir des connexions entre les systèmes individuels en utilisant une variété de type de réseaux: réseau téléphonique, réseau commuté par paquets, réseaux locaux,... . Il permet aux processus d'application ayant établi ces connexions d'échanger des messages de façon fiable entre les systèmes ouverts.

Comme le montre la figure 2.d, deux sous-couches contribuent à l'implantation des fonctions du MHS. Celles-ci sont:

- User Agent Layer ( UAL ) reprenant les

fonctionnalités du UA associées au contenu du message.

- Message Transfer Layer ( MTL ) reprenant des fonctionnalités du MTA et fournissant les services MT qui permettent aux UAs d'accéder et d'être accédés par le MTS en vue d'échanger des messages; outre ces services fondamentaux, la sous-couche MTL en reprend aussi d'autres additionnels tels que la possibilité de livraison différée d'un message, la possibilité de livraison diffusée d'un message, la possibilité de conversion du contenu d'un message,....



UTILISATEUR

UTILISATEUR

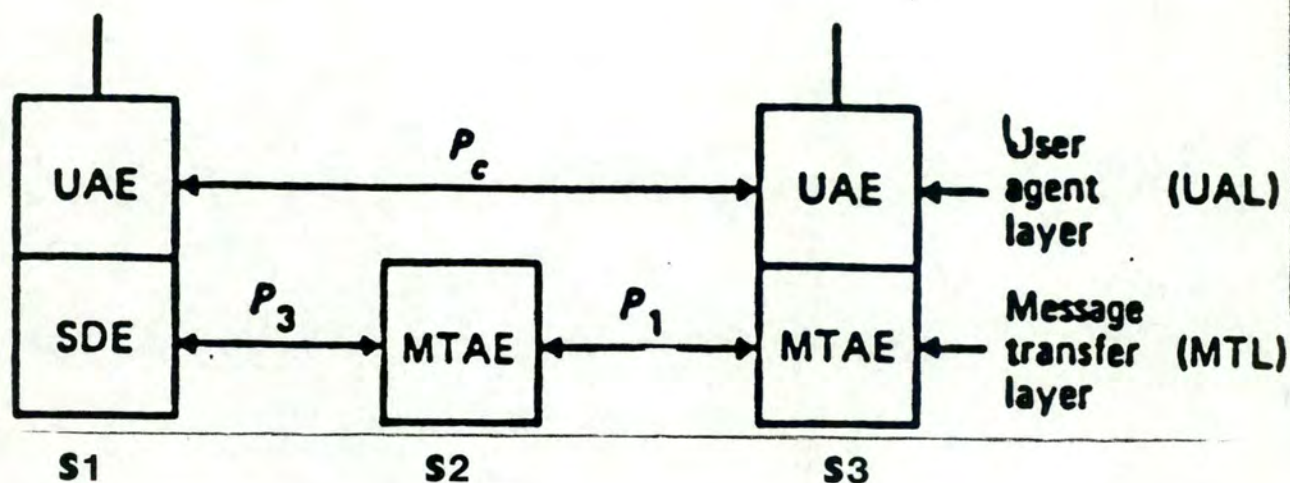


fig 2.d: Le modèle en couches du MHS

En se basant sur les différentes perspectives physiques illustrées précédemment, trois types de système peuvent être distingués dans la figure 2.d:

- Le système S1 fournissant uniquement les fonctions de l'UA
- Le système S2 fournissant uniquement les fonctions du MTA
- Le système S3 fournissant les fonctions de l'UA et du MTA.

Nous voyons apparaître trois types d'entités fonctionnelles réparties dans les deux sous-couches constituant le niveau application du MHS.

La sous-couche UAL abrite l'entité User Agent Entity (UAE). Cette entité renferme les fonctions de l'UA qui

traitent de la représentation du contenu du message. Nous devons, ici, attirer l'attention du lecteur sur une remarque concernant la terminologie. L'UA fait référence aux fonctionnalités représentées par le système S1 tandis que l'UAE est utilisé pour les fonctions strictement associées aux opérations de certains protocoles d'UA à UA.

La sous-couche MTL renferme l'entité Submission and Delivery Entity ( SDE ) et l'entité Message Transfer Agent Entity ( MTAE ).

- L'entité MTAE procure, en coopérant avec d'autres MTAEs, les fonctions requises pour le service de la sous-couche MTL. Cette entité est donc responsable du contrôle de cette sous-couche.

- L'entité SDE est toujours groupée avec un UAE. Il permet, à cet UAE, d'obtenir les services de la sous-couche MTL. Etant responsable des interactions de soumission et de livraison avec un MTAE, il rend accessible au UAE, le MTAE qui lui est associé. Il supporte donc le protocole de communication nécessaire pour qu'un UAE autonome puisse interagir avec un MTAE.

Ces entités communiquent entre elles au moyen de protocoles. Nous en remarquons trois:

- Le protocole P1 de transfert de messages qui définit les relais de ceux-ci entre MTAs ainsi que d'autres interactions nécessaires pour fournir les services MT.

- Le protocole P3 de soumission et de livraison qui réalise la communication entre un SDE et un MTAE. P3 est défini dans la recommandation CCITT X400. IL permet



l'accessibilité des services de la sous-couche MTL à un UAE éloigné. Ce protocole spécifie au bit près la communication entre les sous-couches UAL et MTL. C'est ainsi un cas particulier d'un protocole régissant des échanges d'informations verticaux. Dans le cas de cette configuration physique, l'entité SDE joue le rôle d'entité de jonction entre les deux interlocuteurs.

- Les protocoles Pc d'interaction entre UAEs qui représentent une série de protocoles: chaque instantiation de Pc définissant la syntaxe et la sémantique du contenu des messages transférés, correspond à une classe d'UAEs. La messagerie interpersonnelle en est une et lui est associé le protocole P2.

Bien que la norme X400 ( X400 p.26 à 34 ) ne présente que ce protocole P2, nous pourrions en imaginer d'autres qui permettraient, par exemple, l'échange de messages bancaires.

Dans le chapitre suivant, nous détaillerons les composants de l'entité SDE et nous spécifierons le protocole lui permettant de communiquer avec l'entité MTAE.

-----  
de communication  
-----

Dans ce chapitre, nous détaillerons la structure de l'entité SDE ( Section 1 ) et ensuite nous spécifierons le protocole P3 permettant à cette entité de communiquer avec l'entité MTAE.

3.1. L'entité SDE  
-----

Appartenant à la sous-couche MTL, l'entité SDE supporte le protocole de communication pour qu'une entité UAE autonome puisse communiquer avec un MTAE afin de bénéficier des services de la sous-couche MTL. Cette communication se réalise au moyen d'un ensemble d'opérations ( Submit, Deliver,... ) et est totalement transparente pour l'entité UAE: c'est comme si elle possédait son MTAE local.

Outre l'accès au MTAE, les opérations constituant P3 sont sémantiquement similaires aux opérations permettant à deux MTAEs de communiquer entre elles.

L'entité SDE peut être modélisée comme étant constituée de trois composants hiérarchiquement reliés à chacun des autres.



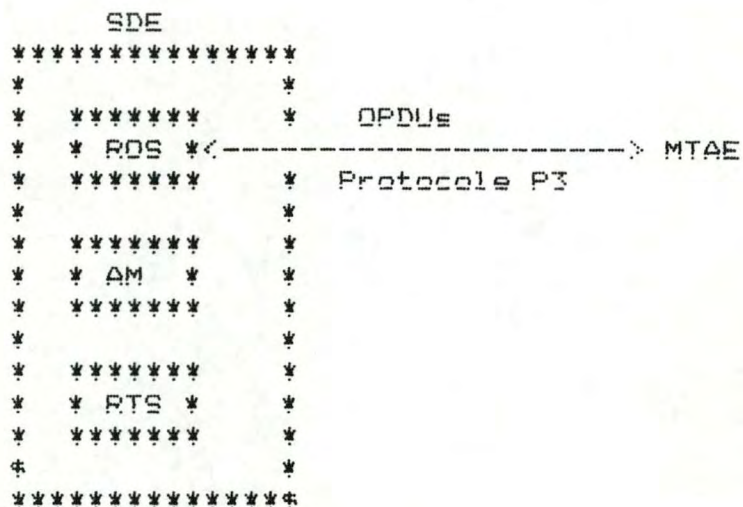


fig 3.2: La structure d'un SDE

Comme nous le montre la figure 3.2, la structure du SDE se compose dans un ordre descendant du:

- Remote Operation Server ( RDS )
- Association Manager ( AM )
- Reliable Transfer Server ( RTS ).

Les fonctions de ces composants sont décrites ci-après.

#### 3.1.1. Le Remote Operation Server ( RDS )

Ce composant RDS renferme le mécanisme de Remote Operation et de Remote Error qui permet la mise en oeuvre d'opérations à distance. ( X410 ).

Le RDS communique avec un autre composant RDS situé dans la structure d'un MTAE en échangeant des Operations Protocole Data Units ( OPDUe ). L'OPDU est une unité d'informations permettant d'une part au SDE de demander des opérations au MTAE et d'autre part au MTAE de

renvoyer au SDE le résultat de l'exécution de l'opération demandée. Cette communication est gérée par un ensemble de règles connu sous le nom de protocole P3 de soumission et livraison: la section 2 de ce chapitre spécifie les opérations définies dans ce protocole.

Le lecteur désireux de s'informer sur le mécanisme du ROS, consultera le chapitre 4 de ce mémoire où cette technique y est détaillée.

### 3.1.2. L'Association Manager ( AM )

-----

Le composant AM intermédiaire est chargé de l'établissement, du contrôle et du relachement des associations logiques en utilisant les primitives de service fournies par le Reliable Transfer Server ( RTS cfr infra ). Il est aussi concerné par l'assignation des OPDUs à ces associations en fonction de certains critères comme le type ou la priorité des OPDUs.

### 3.1.3. Le Reliable Transfer Server ( RTS )

-----

Le composant RTS propose un certain nombre de primitives de service au reste de l'entité qui est dénommé utilisateur-RTS. Ces primitives permettent de créer, de gérer les associations entre deux entités, ainsi que de transférer, avec fiabilité, des unités d'information, appelées Application Protocol Data Units ( APDUs ), entre ces entités au cours d'une association.

Pour réaliser ces services, le RTS fait appel à la



couche session du modèle ISO.

Les primitives sont définies dans la norme X410/84. ( X410 )

### 3.2. Le protocole de soumission et livraison, P3

-----

Lorsque l'UAE et le MTAE résident dans un même système, l'interaction entre ces entités s'effectue directement au travers de la frontière des sous-couches UAL et MTL. En utilisant les fonctions appropriées au travers de la limite séparant ces deux sous-couches, le MTAE fournit ainsi à l'UAE les services MTS.

Par contre dans certaines configurations, ces deux entités ne sont pas groupées physiquement ( cfr fig 2.c ). Pour obtenir les services de la couche MTL, l'UAE, se trouvant sur le site éloigné, doit communiquer avec un MTAE approprié. Cette communication requiert la présence de l'entité SDE. Celle-ci appartenant au système renfermant l'UAE concerné, se localise au niveau de la sous-couche MTL.

Une telle communication a pour but de transférer les messages et la responsabilité de ceux-ci. Elle se décompose en deux interactions: la soumission, lorsque le SDE envoie de l'information au MTAE, et la livraison, lorsque le MTAE envoie de l'information au SDE. Bien que ces deux entités appartiennent à la même sous-couche, il est donc nécessaire que cette communication se déroule au moyen d'un ensemble d'opérations à distance. Ces

opérations sont standardisées dans le protocole P3 de soumission et livraison défini dans la recommandation X411/84. ( X411 ).

Une opération à distance est une opération dont l'exécution est demandée par un site maitre ( le demandeur ) à un site serveur ( l'exécuteur ). Ces deux sites ne sont pas situés dans le meme système informatique.

Le mécanisme d'une opération à distance demande l'apport de quelques détails expliquant son déroulement.

Le site faisant appel à une opération à distance n'est jamais celui qui l'exécute. Il est impératif de décomposer le déroulement de l'opération en primitives appartenant à un type spécifique.

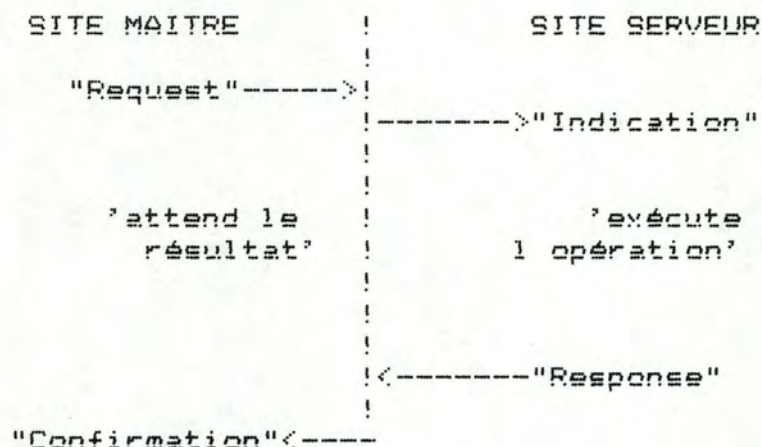


fig 3.b: Le déroulement d'une opération à distance

Comme le montre la figure 3.b, nous distinguons quatre types de primitives:



- la primitive de type " Request " permet au site maitre de demander l'exécution d'une opération à distance.

- la primitive de type " Indication " permet au site serveur d'être averti de la demande d'exécution d'une opération. Il doit tenter de l'exécuter.

- la primitive de type " Response " permet au site serveur de renvoyer le résultat de l'exécution d'une opération qui lui a été précédemment demandée.

- la primitive de type " Confirmation " permet au site maitre d'être averti de la présence du résultat de l'exécution d'une opération à distance qui a été précédemment demandée.

Entre l'envoi d'un " Request " et la réception d'une " Confirmation ", le site maitre est en attente d'une réponse du site serveur. Par contre, ce dernier est actif dans l'intervalle de temps délimité par les primitives " Indication et Response ", durant lequel il tente d'exécuter l'opération qui lui a été demandée.

Pour certaines opérations, il n'est pas nécessaire de transférer le résultat de leur exécution au site maitre; leur déroulement se limite donc aux primitives " Request et Indication ".

Le protocole P3 est donc constitué d'opérations à distance. Elles sont classées en quatre groupes:

- les opérations de gestion
- les opérations de soumission
- l'opération Deliver
- l'opération Notify.

Les services rendus par ces opérations sont conformes avec ceux offerts par la sous-couche MTL à la sous-couche UAL.

### 3.2.1. Les opérations de gestion

-----

Pour que le SDE et le MTAE puissent contrôler le dialogue entre eux, ces deux entités ont besoin d'opérations de gestion qui leur permettent d'échanger de l'information. La gestion de cette communication est réalisée au moyen des opérations Register, Control et Change Password .

#### 3.2.1.1. L'opération Register

-----

L'UA possède des paramètres stockés dans des registres et dont leurs valeurs fixent certaines propriétés telles que le type d'information encodée qui peut être délivrée à l'UA, la longueur maximale de l'information délivrable, le nom E/D de l'UA,... . L'opération Register permet soit au SDE soit au MTAE, d'échanger les valeurs de ces paramètres et ainsi de modifier les propriétés de l'UA. Les primitives Register Request et Register Confirmation correspondent à cette opération.



### 3.2.1.2. L'opération Control

---

Au moyen de l'opération Control, le SDE et le MTAE peuvent restreindre temporellement la possibilité pour l'autre entité de leur demander des opérations. Parmi les restrictions qui peuvent être imposées, nous mentionnerons les suivantes:

- les messages dont la longueur excède une certaine valeur spécifiée, ne peuvent être transférés.
- les messages avec un type spécifié d'informations encodées, peuvent être transférés.
- les messages contenant certains types d'opérations, ( ex: l'opération Notify, l'opération Probe ), peuvent être transférés.

Les primitives Control Request, Control Response, Control Indication, Control Confirmation correspondent à cette opération.

### 3.2.1.3. L'opération Change Password

---

Les entités UAE et MTAE possèdent un mot de passe dont la valeur courante peut être modifiée par l'opération Change Password.

Si l'opération est acceptée par l'entité, la nouvelle valeur de son mot de passe sera celle spécifiée dans l'opération. Dans le cas contraire, elle garde la même valeur pour son mot de passe.

### 3.2.2. Les opérations de soumission

---

Lors de la liaison avec le MTAE, le SDE a la possibilité de lui soumettre des messages. Selon la nature de l'information qu'il veut soumettre, le SDE a le choix entre l'opération Submit, Probe, ou Cancel.

#### 3.2.2.1. L'opération Submit

---

Le SDE peut, au moyen de l'opération Submit, soumettre au MTAE un message complet constitué d'une enveloppe comprenant les valeurs des paramètres nécessaires au transfert du message et du contenu dans lequel se trouve l'information qui doit être transférée.

Le MTAE peut accepter ou refuser l'opération et donc le message.

Selon le cas, un résultat, une indication de rejet ou une spécification de l'erreur rencontrée est envoyé vers le SDE ayant demandé cette opération. Les primitives Submit Request et Submit Confirmation correspondent à l'opération Submit.

#### 3.2.2.2. L'opération Probe

---

L'opération Probe permet au SDE de négocier les paramètres de sa communication avec le MTAE. Les paramètres concernent la longueur du message, le type du



contenu,...

En utilisant cette opération, le SDE peut soumettre au MTAE un message limité comprenant certains paramètres contenus dans l'enveloppe.

Le MTAE accepte ou refuse l'opération qui lui a été soumise et donc l'enveloppe. Si le Probe réussit, le SDE reçoit l'identificateur de l'opération accompagné de l'heure de transfert de celle-ci. Si le Probe échoue, le SDE est averti de la raison pour laquelle le MTAE refuse de communiquer sur base des valeurs des paramètres soumis.

A cette opération correspondent les primitives Probe Request et Probe Confirmation.

#### 3.2.2.3. L'opération Cancel

-----

Certains messages peuvent être soumis au MTAE avec la mention de livraison différée. Tant que les messages n'ont pas quitté le MTAE, le SDE peut demander de les annuler. Le SDE fera appel à l'opération Cancel, et les messages devant être annulés seront spécifiés par l'identificateur de l'opération Submit précédemment réalisée avec succès.

Les primitives Cancel Request et Cancel Confirmation correspondent à cette opération.

#### 3.2.3. L'opération Deliver

-----

L'opération Deliver permet au MTAE de délivrer au SDE

des messages recus.

A cette opération, correspond l'unique primitive Deliver Indication.

#### 3.2.4. L'opération Notify

-----

Au moyen de l'opération Notify, le MTAE peut transférer au SDE des avis de livraison et de non-livraison. Le SDE est alors informé de la réussite ou de l'échec de l'opération de livraison d'un message

Un identificateur d'une opération Submit ou Probe spécifie à quel message le renseignement est relatif.

Aucun résultat n'est renvoyé au MTAE, seulement, la présence d'une erreur peut lui être renvoyée dans le cas où l'opération Notify est en contradiction avec des restrictions pré-établies au moyen de l'opération Control.

La primitive Notify Indication correspond à cette opération.

Après avoir spécifié le protocole P3, nous allons exposer le mécanisme qui permet de réaliser les opérations à distance reprises dans ce protocole.



### 4.0. Introduction

---

Après avoir étudié le SDE, le rôle joué par cette entité, nous allons nous attarder à décrire plus particulièrement l'un de ses composants: le Remote Operation Service ( ROS ).

Le ROS fournit un mécanisme permettant de réaliser des opérations sur un système ouvert et distribué. Ces opérations sont caractérisées par le fait que le système les exécutant est différent et généralement distant de celui ayant demandé l'exécution des opérations. Cette classe d'opérations porte le nom d'opérations à distance. Au cours du déroulement de ces opérations, deux entités, appartenant chacune à un système, sont en interaction. Des structures de données, appelées Remote Operation et Remote Error, servent à modéliser ces opérations à distance.

Celles-ci se composent principalement de deux phases: la phase de demande durant laquelle l'exécution d'une opération est demandée par une entité à une autre avec qui elle s'est préalablement associée, ensuite la phase de réponses durant laquelle l'entité ayant exécuté l'opération, envoie le résultat de l'exécution à l'entité qui l'avait demandé.

Après avoir discerné sur le modèle ROS dans sa généralité, nous examinerons son application de protocole

#### 4.1. Le déroulement d'une opération à distance

-----

Le concept d'opération à distance permet de réaliser de façon concise les protocoles interactifs. La figure 4.a modélise le déroulement d'une opération à distance.

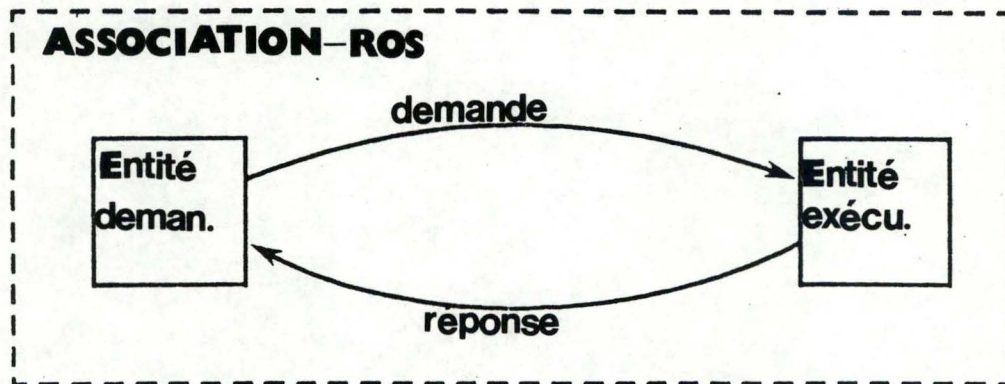


fig 4.a: Le déroulement d'une opération à distance

Le déroulement d'une opération à distance peut être présenté comme une interaction d'éléments atomiques entre les entités liées par une association. Les éléments échangés sont soit des requêtes d'exécution d'une certaine opération sur certains arguments, soit la réponse envoyée par l'entité ayant exécuté l'opération, informant l'entité l'ayant requise du résultat de l'opération. L'association logique durant laquelle se déroule ces échanges est créée par une des deux entités concernées au début de l'interaction et est relâchée lorsque les entités associées n'ont plus lieu de l'être.



#### 4.2. L'aspect fonctionnel du RDS

-----

Le mécanisme offert par le RDS, pour réaliser des opérations à distance, est indépendant de la sémantique des opérations, c'est-à-dire du traitement effectué par ces opérations. Cette abstraction établit une communication entre deux entités que nous appellerons " Utilisateur-RDS " au moyen d'une association appelée " Association-RDS " entre ceux-ci. Ce concept permet à un utilisateur-RDS ( l'appelant ) de commencer et de terminer une association avec un autre utilisateur- RDS ( le répondant ). Une fois l'association-RDS établie, entre deux utilisateurs RDS, le RDS permet à un de ceux-ci de demander l'exécution d'une opération à distance à l'autre utilisateur-RDS. Ce dernier va tenter d'exécuter l'opération requise. Ce mécanisme inclut non seulement le transfert du demandeur vers l'exécuteur de l'opération à effectuer accompagnée des arguments sur lesquels elle doit être exécutée mais aussi l'acheminement de l'information, de l'exécuteur vers le demandeur pour renseigner celui-ci soit du résultat obtenu après l'exécution de l'opération, soit de la présence d'une erreur rencontrée au cours de l'exécution. De façon générale, tant l'appelant que le répondant d'une association-RDS peuvent être, chacun à leur tour, demandeur d'une opération à distance.

Pour compléter cette présentation, nous mentionnerons que des services additionnels de rejet d'une opération sont prévus pour supporter le traitement d'erreur

exceptionnel. Ce service de rejet est utilisé lorsque l'opération est syntaxiquement incorrecte ou lorsque l'opération ne peut être exécutée car l'exécuteur n'a pas les ressources ou lorsque l'opération n'a pas été acheminé correctement.

Le ROS convient pour supporter une large gamme d'applications et, en particulier, il est utilisé pour implanter le protocole de soumission et livraison P3.

#### 4.3. L'environnement ROS dans le modèle OSI

Dans le modèle de référence OSI, le service ROS est fourni par une sous-couche appartenant à la couche application. Comme le montre la figure 4. b. les utilisateurs ROS sont des entités de la couche application ( AE ) se conduisant comme des demandeurs du service ROS

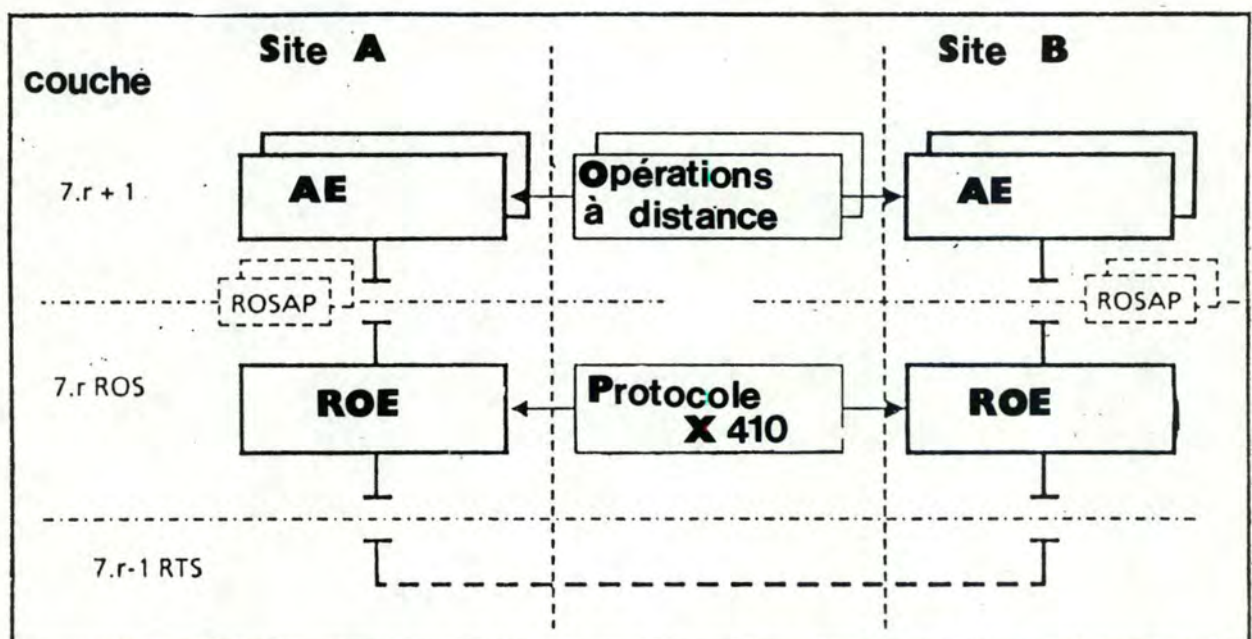


fig 4.b: L'environnement ROS dans le modèle OSI



De leur côté, les entités Remote Operation ( ROE ) jouent le rôle de fournisseurs du service RDS. Les AEs accèdent au RDS en communiquant avec les ROEs par des points d'accès en utilisant des primitives de services. Pour communiquer entre elles, les AEs utilisent un protocole d'application comme le protocole de soumission et livraison tandis que les ROEs se servent du protocole de Remote Operation défini dans la norme CCITT X410. Les ROEs ont besoin des services fournis par les couches et les sous-couches inférieures. Par exemple le Reliable Transfer Server ( RTS ) défini dans la norme CCITT X410 permet de transférer les Operation Protocol Data Units ( OPDUs ) spécifiés par le protocole X410.

#### 4.4. Les classes d'utilisation du RDS

-----

Le RDS est utilisé pour réaliser différents protocoles. Les classes d'utilisation du RDS se répartissent selon les exigences des protocoles que le RDS met en œuvre.

Elles se différencient selon deux critères:

- La classe de l'association-RDS qui est utilisée entre les deux utilisateurs-RDS pour le déroulement d'une

opération à distance. (4.4.1. )

- La classe des opérations à distance qui peuvent être demandées au cours d'une certaine association-RDS (4.4.2. ).

La spécification d'une classe particulière d'utilisation du RDS varie suivant le protocole d'application qui est modélisé ( SIE 85 ).

#### 4.4.1. Les classes d'association-RDS

-----

Rappelons qu'une association-RDS unit toujours deux utilisateurs-RDS qui sont des entités de la couche application ( AE ) . L'AE qui demande une association-RDS est appelée l'appelant tandis que l'AE avec qui l'appelant est associé est appelé le répondant . Au cours d'une association-RDS, une des deux AEs demande une opération à distance et l'autre l'exécute .

La classe de l'association-RDS est déterminée selon la possibilité qu'ont l'appelant et le répondant d'exécuter une opération à distance. C'est ainsi que nous remarquons trois classes d'association-RDS.

- seul le répondant d'une association-RDS peut exécuter des opérations à distance

- seul l'appelant d'une association-RDS peut exécuter des opérations à distance

- l'appelant et le répondant d'une association-RDS peuvent exécuter tous deux des opérations à distance

La figure 4.c, ci-dessous, montre ces trois classes d'association-RDS.



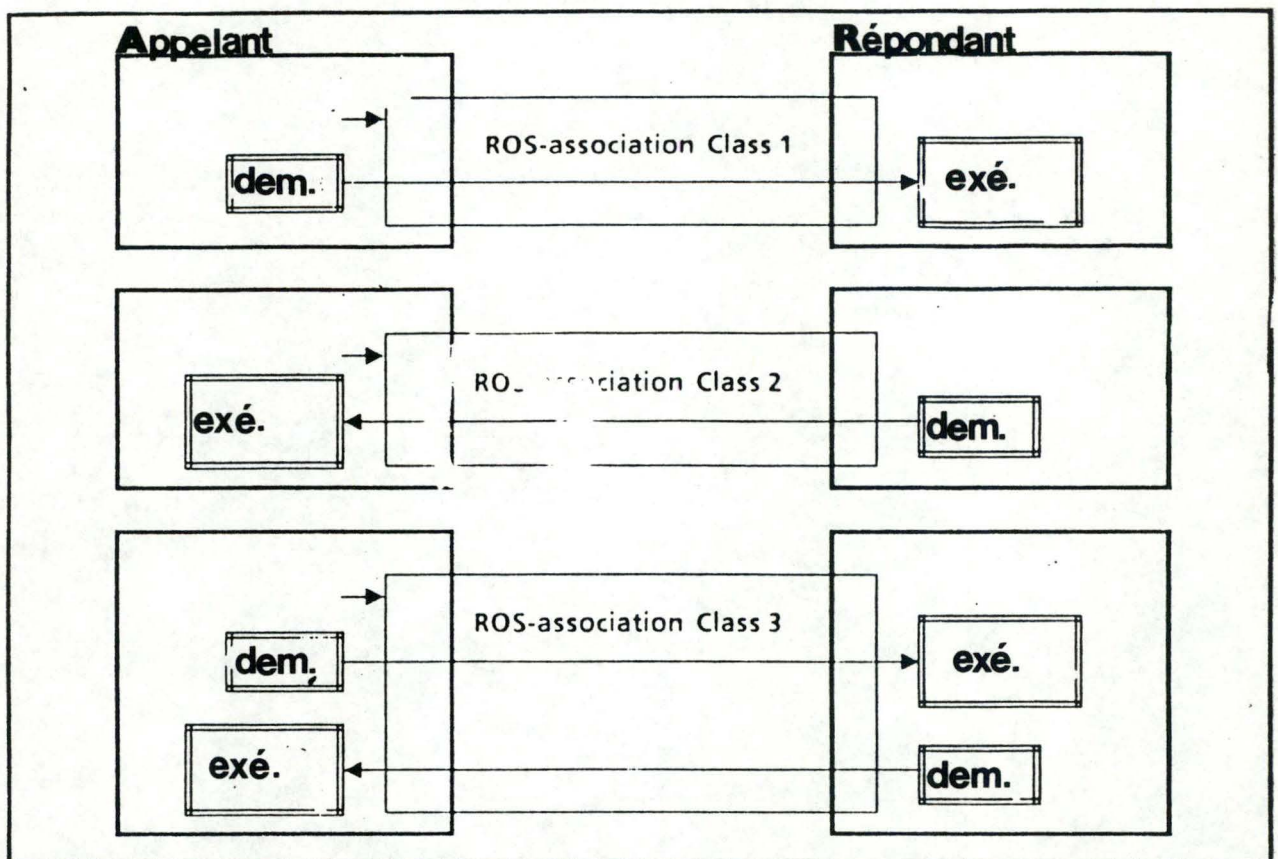


fig 4.c: Les classes d'association-ROS

Pour chaque protocole d'application modélisé par le ROS, une des trois classes d'association-ROS devra être choisie et sera clairement spécifiée.

#### 4.4.2. La classification des opérations à distance

Une opération est caractérisée au moyen de deux critères:

- le type de l'opération
- le mode de l'opération.

#### 4.4.2.1 Le type de l'opération

Le type de l'opération varie, comme le montre la figure 4.d, selon la nature de l'information envoyée, par l'AE exécutant l'opération, à l'AE l'ayant demandée. C'est ainsi que:

- soit un résultat ( opération de type 1 )
  - soit une erreur seulement s'il y en a une ( opération de type 2 )
  - soit aucune information ( opération de type 3 )
- peut être transférée de l'AE exécutant l'opération ( l'exécuteur ) vers l'AE l'ayant invoquée ( le demandeur ).

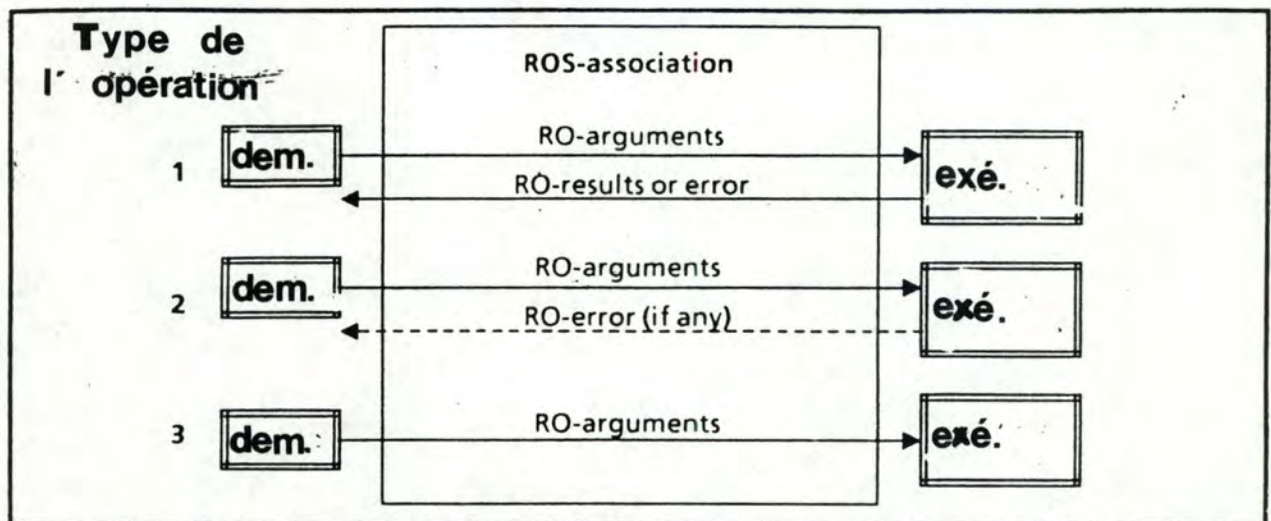


fig 4.d: Les types d'opérations

Dans le cas d'une erreur qui ne survient pas en cours de l'exécution de l'opération mais qui appartient à une autre classe ( ex: une erreur venant du serveur ROS, une erreur venant de l'AE



demandeur ), un service de rejet de l'opération est utilisé pour les trois cas envisagés ci-dessus.

#### 4.4.2.2. Le mode de l'opération

-----

Le mode de l'opération spécifie si l'opération est exécutée soit de façon synchrone ou soit de façon asynchrone.

Le mode de l'opération est soit synchrone quand un AE demandant une opération devra attendre une réponse ( un résultat, une erreur, un rejet ) de la part de l'AE l'exécutant avant de lui demander une nouvelle opération.

Le mode de l'opération est soit asynchrone quand un AE peut demander plusieurs opérations sans attendre d'avoir reçu de l'AE l'exécutant, les réponses correspondantes à chacune de celles-ci. Les réponses peuvent arriver dans un ordre aléatoire ne respectant pas l'ordre de demande d'exécution. C'est par un paramètre identifiant les demandes que le demandeur pourra rétablir la correspondance entre demandeur et répondant pour une même opération.

Si nous combinons de différentes façons ces deux critères, nous définissons plusieurs classes d'opérations.

#### 4.4.3. Les classes d'opérations à distance

-----

Le tableau de la figure 4.c fait apparaître quatre classes d'opérations:

- les opérations renvoyant un résultat de façon synchrone ( classe numéro 1 )
- les opérations renvoyant un résultat de façon asynchrone ( classe numéro 2 )
- les opérations renvoyant une erreur, s'il y en a, de façon asynchrone ( classe numéro 3 )
- les opérations ne renvoyant aucune information et s'exécutant de façon asynchrone ( classe numéro 4 )

| Mode de l' opération | Type de l' opération |            |   |
|----------------------|----------------------|------------|---|
|                      | 1                    | 2          | 3 |
| synchrone            | 1                    | inexistant |   |
| asynchrone           | 2                    | 3          | 4 |

fig 4.e: Les classes d'opérations à distance

Dans un protocole d'application utilisant le RQS, la classe des opérations doit être spécifiée pour chacune d'entre elles. Nous mentionnerons que la classe d'opérations numéro 1 est parfois appelée " appel de procédures à distance " ( RPC ) .



#### 4.4.4. Les combinaisons entre classe d'association-ROS

et classe d'opérations.

Le tableau de la figure 4.f montre les combinaisons possibles entre les classes d'association-ROS et les classes d'opération à distance. Ces différentes combinaisons définissent des classes d'utilisation du ROS.

Le choix sera fait parmi celles-ci et il dépendra de la spécificité du protocole d'application qui utilisera le service ROS.

| Classe de l'opération | Classe d'ass.-ROS<br>opération exé. par |             |                  |
|-----------------------|-----------------------------------------|-------------|------------------|
|                       | rép.<br>1                               | appel.<br>2 | les<br>deux<br>3 |
| classe 1              | a                                       |             |                  |
| classe 2              |                                         |             | b                |
| classe 3              |                                         |             |                  |
| classe 4              |                                         |             | b                |

fig 4.f: Les classes d'utilisation du ROS

La combinaison, dans la figure 4.c, de la classe d'association-ROS numéro 1 et de la classe d'opération



numéro 1 convient pour les protocoles d'accès dans les stations de travail ( ex: P7, le protocole d'accès à la boîte aux lettres ).

La combinaison ( b ) de la classe d'opération numéro 2 et numéro 4 nous concerne plus particulièrement pour ce rapport car elle est utilisée dans le service de transfert du courrier pour structurer le protocole P3 de soumission et livraison.

#### 4.5. Les concepts de Remote Operation et

##### Remote Error

La spécification concise et la mise en oeuvre de protocoles interactifs comme le protocole de soumission et livraison, sont rendues plus aisées en utilisant les concepts de Remote Operation et de Remote Error aussi appelés respectivement Operation et Error. Pour définir ces concepts nous ferons appel aux macros. Les macros ( X409 p.68 et 69 ) définissent un type non standard. Elles permettent aux programmeurs de définir ses propres types en utilisant des types définis comme le type entier, le type réel, le type booléen,... . A chaque macro est attribué un nom en lettres capitales; celui-ci référence le type construit.

Pour les lecteurs désirant obtenir davantage d'informations sur les macros, ils pourront consulter la recommandation X409. ( X409 )



#### 4.5.1. Le concept de Remote Operation

-----

Une Remote Operation représente une structure qui détermine la spécification du déroulement d'une opération à distance dont une AE peut demander l'exécution à une autre AE.

Cette structure renferme trois champs dont leur présence est facultative et dépend du type de l'opération spécifiée. Ces champs sont les suivants:

- le champ Argument: il spécifie le paramètre argument et son type. Ce champ est présent si l'opération définie porte sur un argument, si non il est absent.

- le champ Résultat: il spécifie le paramètre résultat et son type. Ce champ est présent si l'opération définie requiert le retour d'un résultat suite à son exécution, si non il est absent

- le champ Erreur: il spécifie une liste d'erreurs qui pourraient survenir lors de l'exécution de l'opération définie. Chaque élément de cette liste renvoie à une structure de Remote Error ( cfr. 4.5.2. ). Si aucune erreur ne peut être rencontrée, ce champ est absent.

Ainsi une opération de type 2 ( cfr. 4.4.2.1. ) portant sur un argument sera représentée par une structure renfermant le champ Argument et le champ Erreur.

Cette structure est décrite au moyen de la syntaxe des macros et il lui est assigné le mot-clé OPERATION comme nom de référence.

A toute structure de type OPERATION est attribuée une

variable numérique dont la valeur identifie l'opération définie par cette structure.

#### 4.5.2. Le concept de Remote Error

-----

Une Remote Error représente une structure déterminant la spécification d'une erreur qui peut être renvoyée par l'AE exécuteur à l'AE demandeur en conséquence de l'exécution d'une opération. Cette erreur correspond à un ensemble de conditions exceptionnelles pouvant être rencontrées par l'AE exécuteur lorsqu'il exécute l'opération.

Cette structure se compose d'un seul champ: le paramètre qui permet d'énumérer l'ensemble des conditions relatives à l'erreur définie par cette structure.

La structure de Remote Error est décrite au moyen de la syntaxe des macros et il lui est assigné le mot-clé ERRDR comme nom de référence.

A toute structure de type ERRDR est attribué une variable numérique dont la valeur identifie l'erreur définie par cette structure.

Les structures définissant des Remote Operation et des Remote Error sont connues de l'exécuteur et du demandeur d'une opération. Afin de réaliser ces opérations, ces structures sont décomposées en unités d'information appelées Operation Protocol Data Units ( OPDUs ).



#### 4.6. Les types d'OPDU

---

Quand une AE demande l'exécution d'une opération à une autre AE, cette exécution à distance s'effectue par un échange d'Operation Protocol Data Units ( OPDU ) entre ces deux AEs. Il existe quatre types d'OPDU. Chacun d'entre eux remplit une fonction bien précise:

- l'OPDU de demande invoque l'exécution d'une opération.
- l'OPDU de résultat rapporte le résultat d'une opération s'étant déroulée avec succès.
- l'OPDU d'erreur rapporte l'erreur rencontrée lors de l'exécution d'une opération.
- l'OPDU de rejet renvoie un renseignement informant que l'OPDU reçu est malformé.

##### 4.6.1. L'OPDU de demande

---

Un OPDU de demande a pour but de demander l'exécution d'une opération à distance. Il est envoyé par un AE lorsque celui-ci désire qu'un autre AE exécute cette opération. Il est constitué de trois composants:

- L'identificateur de demande permettant de spécifier l'OPDU afin de pouvoir mettre en rapport les réponses conséquentes à l'exécution de l'opération avec la demande
- L'identificateur de l'opération permettant de spécifier l'opération qui doit être exécutée à distance
- L'argument permettant de spécifier la donnée sur laquelle l'opération doit être exécutée.

Un AE demandeur peut donc générer un OPDU de demande disant: " Voici la demande numéro X, exécuter l'opération numéro Y en utilisant l'argument suivant en entrée, Z " ( Z est l'argument en entrée dont le type est spécifié par la macro définissant l'opération Y ). L'OPDU de demande sera donc la suivante:

DEMANDE ( X, Y, Z ).

#### 4.6.2. L'OPDU de résultat

-----

Un OPDU de résultat a pour but de renvoyer à l'AE ayant demandé une opération le résultat de celle-ci. Donc un OPDU de ce type est une réponse à un OPDU de demande si l'opération réussit. L'OPDU de résultat est constitué de deux composants:

- L'identificateur de demande permettant de spécifier l'OPDU de demande qui a servi à l'AE demandeur pour invoquer l'opération à distance dont ce résultat est conséquent. Ce composant met ainsi en rapport demande et résultat pour une opération

- Le résultat permettant de spécifier la donnée qui est le résultat de l'exécution de l'opération demandée.

Après avoir exécuté une opération sans avoir rencontré d'erreur, comme spécifié dans la macro, un AE serveur génère un OPDU de résultat en disant: " Voici la conséquence de la demande numéro X pour cette session, T " ( T est le résultat dont le type est spécifié dans la macro définissant l'opération invoquée par l'OPDU de demande numéro X ). L'OPDU résultat sera donc:

RESULT ( X, T ).



#### 4.6.3. L'OPDU d'erreur

-----

Un OPDU d'erreur a pour but de renvoyer à l'AE ayant demandé une opération à distance, une indication que celle-ci n'a pu être réalisée: il y a une erreur contredisant les conditions d'exécution spécifiées dans la macro définissant cette opération. Donc lorsque l'opération ne peut être exécutée, l'AE exécuteur répond à un OPDU de demande par ce type d'OPDU. L'OPDU d'erreur est constitué de trois composants:

- L'identificateur de demande permettant de spécifier la demande à laquelle fait référence cet OPDU afin de mettre en rapport l'erreur avec une opération préalablement invoquée ( cfr identificateur de demande dans l'OPDU de résultat ).

- L'identificateur d'erreur permettant de spécifier l'erreur rencontrée et prévue par la macro définissant l'opération. Cet identificateur fait référence à une macro définissant l'erreur.

- Le paramètre permettant de spécifier parmi les possibilités d'échec reprises dans la macro définissant l'erreur, laquelle est rencontrée lorsque l'AE exécuteur essaie d'exécuter l'opération demandée. Le type de ce composant est spécifié par la macro qui définit l'erreur.

Lors de l'exécution d'une opération à distance, l'AE exécuteur, s'il rencontre une erreur prévue par la macro définissant cette opération, génère un OPDU d'erreur en disant: " Voici la conséquence de la demande numéro X, pour cette session, l'erreur numéro R a été rencontrée et

le paramètre à la valeur param. " ( le type de la valeur param. est spécifié dans la macro définissant l'erreur numéro R ). L'OPDU d'erreur sera donc :

ERREUR ( X, R, param. ).

#### 4.6.4. L'OPDU de rejet

-----

Un OPDU de rejet a pour but d'informer un AE qu'un OPDU préalablement envoyé n'est pas accepté. Un OPDU est reçu par le correspondant mais celui-ci ne peut l'accepter car il est malformé. Un OPDU de demande, de résultat ou d'erreur peut être ainsi rejeté.

L'OPDU de rejet est constitué de composants :

- l'identificateur de demande permettant de spécifier l'OPDU qui est rejeté. La valeur de ce composant peut être absente dans le cas où l'OPDU reçu est malformé et ne possède pas d'identificateur. Il prend alors la valeur NULL. Dans les autres cas, sa valeur est égale à celle de l'identificateur de demande de l'OPDU qui est à rejeter. Le composant est du type entier

- le problème rencontré permettant de spécifier la raison pour laquelle l'OPDU a été rejeté. Tous les problèmes pouvant être rencontrés sont classés ci-dessous et sont classés en quatre catégories :

- a) problèmes généraux : cette rubrique reprend trois raisons de rejet de l'OPDU dont le type n'est pas déterminant :

- le type de l'OPDU reçu n'est pas reconnu car il n'appartient à aucun des quatre types spécifiés dans la recommandation X410/84.



-- la structure de l'OPDU reçu ne coïncide pas avec celles définies dans la recommandation X409/84.

b) problèmes spécifiques à un OPDU de demande: cette rubrique reprend les quatre raisons de rejet d'OPDU de demande.

-- la valeur de l'identificateur de l'OPDU reçu est un doublon: elle contredit la condition imposant qu'à chaque OPDU ne corresponde qu'une et une seule valeur de l'identificateur

-- l'opération à distance demandée par l'OPDU n'est définie par aucune structure de Remote Operation. L'AE recevant l'OPDU ne connaît pas l'opération dont l'exécution lui est demandée

-- le type de l'argument de l'opération ne correspond à aucun type possible, repris dans la structure, Remote Operation, définissant cette opération

-- les ressources dont dispose l'AE devant exécuter l'opération ne sont pas suffisantes pour son exécution: l'OPDU est alors rejeté.

c) problèmes spécifiques à un OPDU de résultat: cette rubrique reprend les trois raisons de rejet d'un OPDU de résultat:

-- la valeur de l'identificateur de l'OPDU reçu ne correspond à aucun OPDU de demande envoyé par l'AE

-- la valeur de l'identificateur de l'OPDU reçu correspond à un OPDU de demande préalablement envoyé. Cependant, selon la structure Remote Operation, définissant cette opération, celle-ci ne prévoit pas le retour d'un résultat. L'OPDU n'a pas lieu d'exister, il

est rejeté.

-- le type du résultat reçu ne correspond à aucun type prédéfini dans la structure, Remote Operation, définissant l'opération.

d) problèmes spécifiques à un OPDU d'erreur: cette rubrique reprend les cinq raisons de rejet d'un OPDU d'erreur:

-- l'identificateur de l'OPDU reçu ne correspond à aucun OPDU de demande envoyée par l'AE

-- la valeur de l'identificateur de l'OPDU reçu correspond à un OPDU de demande préalablement envoyé. Cependant, selon la structure, Remote Operation, définissant cette opération, celle-ci ne prévoit pas le retour d'une erreur. L'OPDU reçu n'a pas de sens: il est rejeté.

-- la valeur reçue de l'identificateur de l'erreur ne correspond à aucune structure, Remote Error, définie entre ces AEs

-- l'erreur reçue n'est pas reprise dans la structure, Remote Operation, définissant l'opération préalablement demandée.

-- le type du paramètre de l'erreur ne correspond à aucun type pré-établi dans la structure, Remote Error, définissant cette erreur.



proposé par SIEMENS

---

## 5.0. Introduction

---

Afin de réaliser notre mémoire de fin d'étude, la société SIEMENS SOFTWARE nous a accueilli à Munich pour effectuer un travail sur le système de courrier électronique qu'elle implante. Il porte le nom de Elektronische Mitteilungssystem ( EM ). Il respecte la normalisation imposée par l'organisme European Computer Manufacturer Association ( ECMA ) sous l'égide du CCITT. L'EM sera ainsi compatible avec des produits proposés par d'autres firmes dans ce domaine.

Au cours de ce chapitre, nous présenterons l'architecture générale de l'EM de SIEMENS. Le modèle TPDU fait l'objet de quelques explications supplémentaires; nous mentionnerons ses caractéristiques propres à l'environnement SIEMENS.

## 5.1. L'architecture générale du EM de SIEMENS

---

La société SIEMENS travaille essentiellement sur deux gammes de machines: le gros ordinateur avec son système d'exploitation BS2000 ( Bedien system ) et le PC multi-tache avec l'environnement SINIX appartenant à la famille UNIX.

Le système EM sera compatible avec ces deux gammes d'ordinateurs.

Tout processus peut traiter des données situées dans son répertoire local en utilisant des fonctions offertes par son environnement de travail. Il peut également envoyer des messages à destination d'autres processus résidents sur le même site. Il peut enfin préparer des messages, des requêtes d'opérations à distance à l'attention de processus sur des sites éloignés, extérieurs à son cadre de travail.

Pour cette dernière tâche, ce processus requiert l'aide de services lui permettant de préparer son message et ensuite de le transférer, avec fiabilité, vers les destinataires concernés.

Dans le schéma général présenté à la figure 5.a, nous découvrons la découpe modulaire proposée chez SIEMENS pour son EM. La hiérarchie des composants s'accompagne du flux de l'information. Les interfaces permettent d'établir une correspondance syntaxique entre les différents modules.

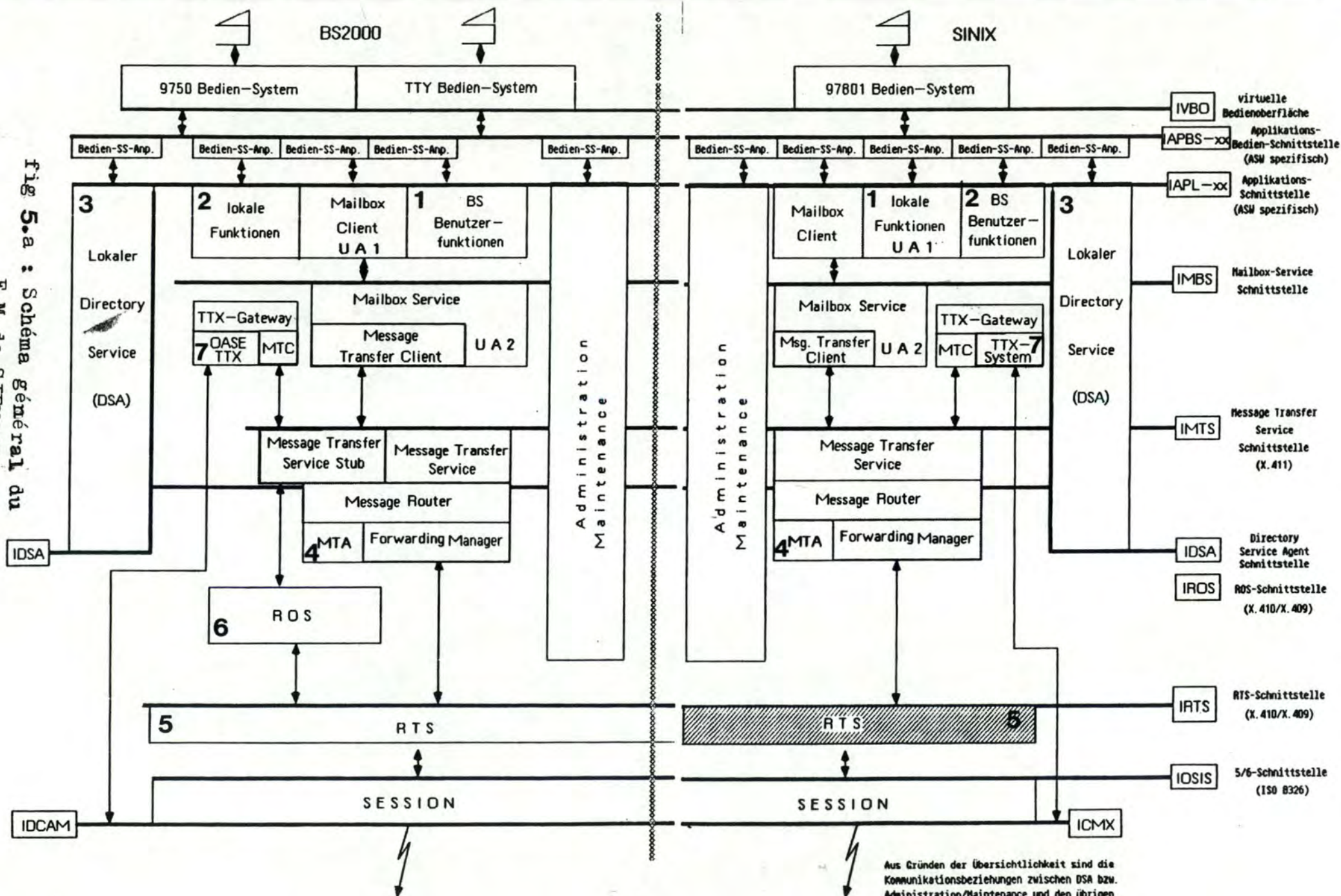
Le service local du site sur lequel travaille le processus est assuré par les modules notés 1, 2 et 3 ( cfr fig 5.a ).

Les services de courrier électronique sont fournis par des modules situés dans la couche application du modèle OSI.

L'entité MTA ( 4 cfr fig 5.a ) permet l'acheminement fiable du message: elle utilise les services fournis par le RTS ( 5 cfr fig 5.a ).



fig 5.a : Schéma général du  
E.M de SIEMENS  
70





Le module RDS ( 6 cfr fig 5.2 ) n'est présent que dans une architecture implantée sur un environnement BS2000 ( moitié gauche de la fig 5.2 ) : en effet, dans cet environnement certains processus n'ont pas accès à un MTA local, il est alors nécessaire d'utiliser le RDS pour communiquer avec un MTA sur un site éloigné. Par contre, dans le cas d'un environnement SINIX, à chaque UA est associé un MTA.

Il est à noter que la version actuelle du RDS qui est implanté ( RDS 86 ), est conforme aux exigences de la norme X410. Elle ne supporte cependant que les associations de la classe numéro 1 ( seul l'appelant d'une association peut être demandeur d'une opération ) et les opérations de la classe numéro 2 ( les opérations renvoient les résultats de façon asynchrone ).

L'utilisation d'une passerelle est parfois nécessaire dans les cas où d'autres méthodes ( ex: TTX ) sont choisis pour transférer le message.

Après cette présentation générale, nous allons citer les caractéristiques du module TPDU chez SIEMENS.

## 5.2. Les caractéristiques du module TPDU

-----Avant de poursuivre ce mémoire, il importe d'attirer l'attention du lecteur sur le point suivant: l'abréviation TPDU fait référence au module Transfer Protocol Data Unit implanté dans l'architecture du courrier électronique de SIEMENS, et non pas à l'unité de transfert portant le même nom et définie dans la norme X411 ( X411 ).



Dans cette section, nous allons présenter brièvement le module TPDU: nous exposerons sa fonction, son environnement au sein de l'architecture EM et nous justifierons sa présence. ( TPDU 86 ).

#### 5.2.1. La fonction du module TPDU

-----

La fonction du module TPDU est de manipuler les fichiers. La méthode d'accès de ces fichiers est restreinte à l'accès séquentiel.

Les opérations que le TPDU peut effectuer sur ces fichiers sont soit:

- ouvrir un fichier séquentiel
- fermer un fichier séquentiel
- écrire dans un fichier séquentiel
- lire à partir d'un fichier séquentiel
- positionner le pointeur d'un fichier séquentiel: les fichiers traités possèdent un pointeur; ce repère logique indique une position ( en byte à partir du début du fichier ) à partir de laquelle toute opération d'entrée/sortie sera réalisée sur ce fichier. Cette opération permet de positionner ce repère
- détruire le fichier séquentiel d'une partie de celui-ci.

### 5.2.2. L'environnement du TPDU

---

Le module TPDU est utilisé par le module ROS lorsque celui-ci a besoin d'opérations qui manipulent des fichiers. Si dans certaines opérations à distance, un traitement est demandé sur un fichier, le ROS fera donc appel au TPDU pour effectuer cette opération. La figure 5.b montre l'environnement du TPDU.

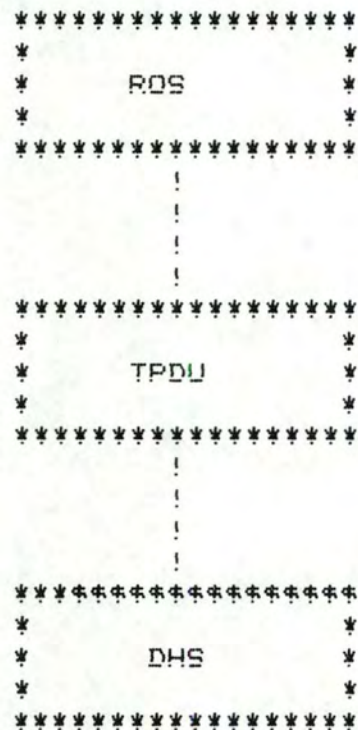


fig 5.b: L'environnement du TPDU



### 5.2.3. La justification du TPDU

---

Pour le RDS, il est important de connaître exactement l'erreur survenue lorsque l'exécution d'une opération échoue. Cependant les appel-systèmes de l'environnement SINIX ne fournissent aucune information détaillée lorsque l'exécution de ceux-ci ne réussit pas. Il est donc nécessaire de compléter ces appels par une analyse préalable des valeurs des paramètres passés en entrée à ces appels. Cette analyse révèle l'existence d'une erreur syntaxique ou sémantique pour ces paramètres. L'erreur peut ainsi être détaillée.

Toute cette analyse justifie donc l'existence d'un module. Elle est développée dans le module spécifique TPDU.

Le chapitre 6 présente le travail qui a pour but de transformer ce module TPDU.

## 6.1. La présentation générale du travail

---

### 6.1.1. Le but du travail

---

Au cours de notre stage chez SIEMENS SOFTWARE à Munich, nous avons réalisé un travail. Celui-ci se situe au coeur de l'architecture du système de courrier électronique développée par cette firme. La présentation de ce système a fait l'objet du chapitre précédent.

Ce travail avait pour but, comme le montre la figure 6.a, de transformer le module TPDU qui offre un ensemble d'opérations permettant de manipuler les fichiers séquentiels.

Pour offrir ce service, le TPDU utilisait le module Data Handling System ( DHS ) qui travaille sur ces fichiers au moyen de fonctions offertes par la bibliothèque du système d'exploitation SINIX.

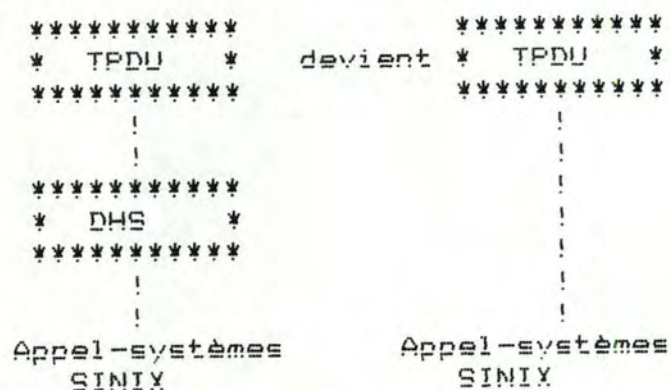


fig 6.a: La présentation générale du travail.



Après avoir effectué cette substitution, le nouveau module TPDU, ainsi obtenu, aura la même spécification qu'auparavant ( cfr section 5.2. ), mais manipulera les fichiers directement au moyen des fonctions de la bibliothèque SINIX. Le travail a donc pour but de court-circuiter l'utilisation du module DHS par le module TPDU.

#### 6.1.2. La justification du travail

-----

Nous faisons remarquer que le nouveau TPDU a un caractère moins structuré que l'ancien. En effet, dans le TPDU de l'ancienne version, les fonctions de la bibliothèque SINIX sont isolées et regroupées

dans le module DHS. Par contre, le TPDU modifié mélange ces fonctions avec d'autres n'étant pas directement issues de la bibliothèque SINIX. Il en résulte une moins grande portabilité du nouveau produit sur différents systèmes d'exploitation.

Cependant, malgré cet inconvénient de non-portabilité, la justification de l'utilité du nouveau TPDU réside dans la politique pratiquée par SIEMENS visant à l'intégration à ses deux environnements: SINIX, BS2000 ( cfr section 5.1. ) de cette architecture de courrier électronique. Dans le cadre de cette réalisation, ils ont créé un logiciel permettant de déterminer automatiquement le caractère de compatibilité aux différents environnements de toutes les fonctions employées dans un module. Afin de

procéder à ce contrôle de conformité pour le module TPDU, il était nécessaire de réaliser la substitution pour centraliser dans un seul module toutes les fonctions auxquelles celui-ci a recours.

Un autre avantage du travail serait un gain de performance lors de l'exécution des opérations offertes par le module TPDU. En effet, le TPDU substitué supprime le niveau du module DHS dans la hiérarchie physique.

#### 6.1.3. Les étapes du travail

-----

Afin de réaliser la tâche qui nous a été confiée, nous avons décidé de la découper en étapes conformément aux théories de méthodologie qui nous ont été enseignées.

Nous avons d'abord procédé à l'analyse du TPDU de l'ancienne version ( section 6.2. ). Elle nous a permis de dégager un certain nombre d'opérations réalisées dans le module.

Cette étape nous a amené à spécifier chaque fonction associée à une opération en terme de paramètres en entrée, en terme de paramètres en sortie, en terme de traitement ( section 6.3. ). Cette étape se fait en étant conscient des outils ( définition de structures de données complexes avec la technique de pointeurs, possibilité d'appeler les fonctions de manipulation de fichiers reprises dans la bibliothèque du système SINIX,... ) que nous offre le langage de programmation " C ", qui nous sert à implanter le travail ( Geh 85 et Kel 84 ).



Nous avons ensuite spécifié de façon interne chaque fonction ( section 6.4. ).

Après avoir implanté les fonctions sur le système SINIX en langage " C ", nous les avons testées. Pour cette étape, nous avons retravaillé le programme de test qui avait servi pour le contrôle de l'ancienne version du module TPDU. Nous avons donc repris les memes jeux de test ( section 6.5. ).

Pour ce qui concerne la documentation du programme, nous l'avons insérée dans le programme lui-meme. Cette étape s'est réalisée conformément aux méthodes de documentation utilisées chez SIEMENS ( cfr listing du programme en annexe ).

## 6.2. L'analyse du TPDU

-----

### 6.2.1. La définition des opérations

-----

Le module TPDU offre un ensemble d'opérations permettant de manipuler des fichiers à accès séquentiel. Ces opérations sont les suivantes:

- ouvrir un fichier séquentiel désigné par son nom d'accès
- lire, à partir d'un fichier séquentiel désigné par son identificateur, un nombre de bytes et les mettre en mémoire centrale dans un tampon
- écrire, dans un fichier séquentiel désigné par son identificateur, un nombre de bytes situées en mémoire

centrale dans une zone tampon

- positionner le pointeur d'un fichier séquentiel désigné par son identificateur à une distance évaluée en bytes à partir du début du fichier

- détruire une partie d'un fichier séquentiel désigné par son identificateur

- former un fichier séquentiel désigné par son identificateur.

Etant donné le caractère atomique de ces opérations, nous avons décidé de faire correspondre, pour chacune d'entre elles, une fonction réalisant le traitement demandé.

#### 6.2.2. La structure de liste

-----

Par apport à l'utilisation directe des fonctions adéquates de la bibliothèque SINIX, pour réaliser les opérations sur les fichiers, le module TPDU permet un contrôle préalable sur les paramètres en entrée ( cfr section 5.2.3. ). Cette vérification fournit des renseignements précis sur les différents cas d'erreur lorsque l'exécution de l'opération échoue.

Afin de vérifier les valeurs des paramètres en entrée de chaque fonction, il est nécessaire de gérer une structure de données en annexe. Celle-ci renferme, à chaque instant, une série d'informations relatives aux différents fichiers traités par le module TPDU. Cette structure est implantée en " C " par le mécanisme de liste chaînée ( Kel 84 p. 280 à 300 ). Chaque élément de



la liste est une structure ( au sens du " C " ) et elle comporte les champs suivants:

- le nom du fichier ( char \* filename ); ce champ est un pointeur vers une chaîne de caractères renfermant le nom d'accès complet du fichier.

- l'identificateur du fichier ( short tpfileid ); ce champ renferme l'identifiant du fichier attribué lors de l'opération d'ouverture.

- le mode d'exploitation du fichier ( Tpdir modus ); ce champ dont le type est construit par le programmeur peut prendre la valeur TPIN si le fichier est ouvert pour réaliser des opérations de lecture. Il prend la valeur TPOUT pour une ouverture en écriture

- la position du pointeur de fichier ( long datalength ); cette position est donnée en bytes et relativement au début du fichier.

- le dernier déplacement du pointeur de fichier ( long offset ); cette distance représente, en bytes, le déplacement du pointeur du fichier suite à la dernière opération sur ce fichier.

- la longueur du fichier en bytes ( long filelength ).

Pour manipuler cette liste, une série de fonctions nous sont offertes; en voici les spécifications externes:

- rt4\_611\_createlisthead: cette fonction crée une tête de liste

- rt4\_612\_create\_and\_append: cette fonction crée un nouvel élément et le rattache en queue de liste

- rt4\_615\_delete\_any: cette fonction enlève de la liste un élément déterminé par un paramètre en entrée.

- rt4\_614\_ seerch: cette fonction recherche dans la liste déterminée par son pointeur de tete, un élément spécifié par l'identificateur du fichier qui lui correspond.

### 6.3. La spécification externe des fonctions du TPDU

-----

Le module TPDU se compose de six fonctions. Pour chacune d'entre elles, nous donnerons la spécification externe.

#### 6.3.1. La fonction tp2\_001\_open

-----

tpopenid = f ( tp\_data\_description, tpdire )

La fonction tp2\_001\_open a pour but d'ouvrir un fichier séquentiel. L'analyse de ses paramètres est la suivante:

Les paramètres en entrée sont :

- tp\_data\_description: une variable de type pointeur vers une chaîne de caractères dont la valeur spécifie le nom d'accès du fichier (Char \*tp\_data\_description )

- tpdire: une variable de type pointeur vers une variable de type défini par le programmeur. Elle spécifie si le fichier est ouvert en lecture ou en écriture. ( Tpdire \*tpdire )

Le paramètre en sortie est:

- tpopenid: une variable de type pointeur vers un short. Si la fonction a réussi ( ret\_code = 0 ), elle contient l'identificateur du fichier renvoyé par la



fonction `open` du système SINIX. Le fichier est alors ouvert. Dans les autres cas, la valeur de cette variable est indéfinie. ( `Short *tpopenid` )

#### 6.3.2. La fonction `tp2_002_read`

-----  
( `tpread_data_length`, `tpinfurther_data` ) = `f`(  
`tpopenid`, `tpinrequired_data_length`, `tpinbufferaddress` )

La fonction `tp2_002_read` a pour but de lire à partir d'un fichier séquentiel, un nombre de bytes et de les ranger en mémoire centrale. L'analyse de ses paramètres est la suivante:

Les paramètres en entrée sont:

- `tpopenid`: une variable de type pointeur vers un `short` dont la valeur identifie un fichier ouvert et sur lequel l'opération de lecture doit être réalisée ( `Short *tpopenid` )

- `tp_inrequired_data_length`: une variable de type pointeur vers un `Short` dont la valeur détermine le nombre de bytes qui doivent être lus ( `Short *tpinrequired_data_length` )

- `tpinbufferaddress`: une variable de type pointeur vers un `char` dont la valeur indique l'adresse de la zone en mémoire où se rangeront les bytes lus. ( `Char *tpinbufferaddress` )

Les paramètres en sortie sont:

- `tp_read_data_length`: une variable de type pointeur vers un `short` dont la valeur indique le nombre de bytes qui ont été effectivement lus ( `Short *tpread_data_length` )

)

- `tp_infurther_data`: une variable de type pointeur vers un booléen dont la valeur est égale à faux si le pointeur du fichier pointe le dernier byte de celui-ci. Dans ce cas, il n'y a plus de données à lire dans le fichier. La valeur est égale à vrai dans les autres cas. Ce paramètre est équivalent à la fonction `eof` du Pascal ( `Bool *tpinfurther_data` ).

Si l'exécution de la fonction `SINIX read` échoue, les valeurs des paramètres en sortie ne sont pas définies.

#### 6.3.3. La fonction `tp2_003_write`

-----  
( `tpwritten_data_length`, `tpoutfurtherdataspace` ) = f ( `tpopenid`, `tpoutrequired_data_length`, `tpoutbufferaddress` )

La fonction `tp2_003_write` a pour but d'écrire dans un fichier séquentiel un certain nombre de bytes rangés en mémoire centrale dans un tampon. L'analyse de ses paramètres est la suivante:

Les paramètres en entrée sont:

- `tpopenid`: une variable de type pointeur vers un short dont la valeur identifie un fichier ouvert sur lequel l'opération d'écriture doit être réalisée ( `Short *tpopenid` )

- `tpoutrequired_data_length`: une variable de type pointeur vers un short dont la valeur détermine le nombre de bytes qui doivent être écrits dans le fichier ( `Short *tpoutrequired_data_length` )

- `tpoutbufferaddress`: une variable de type pointeur



vers un char dont la valeur indique l'adresse en mémoire centrale du tampon où est rangée l'information qu'il faut écrire sur le fichier ( Char \*tpoutbufferaddress )

Les paramètres en sortie sont:

- tpwritten\_data\_length: une variable de type pointeur vers un short dont la valeur indique le nombre de bytes qui ont réellement été écrits dans le fichier ( Short \*tpwritten\_data\_length )

-tpoutfurtherdataspace: une variable de type pointeur vers un booléen dont la valeur est faussé lorsque l'égalité  $tpwritten\_data\_length = tpoutrequired\_data\_length$  n'est pas vérifiée. Toute l'information n'a pu être écrite sur le fichier. Il n'y a donc plus de place pour écrire sur ce support. Dans les autres cas, la valeur de ce paramètre est égale à vraie. ( Bool \*tpoutfurtherdataspace )

Les valeurs des paramètres en sortie ne sont définies uniquement lorsque l'exécution de la fonction SINIX write réussit.

#### 6.3.4. La fonction tp2\_004\_position

-----

--- = f ( tpopend, tpooffset )

La fonction tp2\_004\_position a pour but de positionner le pointeur d'un fichier séquentiel à une certaine distance, évaluée en bytes, à partir du début de celui-ci. Rappelons que le concept de pointeur associé à un fichier rend possible l'exécution d'opérations d'entrée/sortie sur ce fichier à un endroit quelconque et

pouvant être différent du début du fichier. Cette fonction permet d'effectuer le choix de cet endroit. L'analyse de ses paramètres est la suivante:

Les paramètres en entrée sont:

- `tpopenid`: une variable de type pointeur vers un Short dont la valeur identifie un fichier ouvert pour lequel le pointeur doit être positionner à un certain endroit (`Short *tpopenid` )

- `tpoffset`: une variable de type pointeur vers un long dont la valeur détermine en bytes à partir du début du fichier la nouvelle position que devra prendre le pointeur du fichier (`long *tpoffset` ).

Cette fonction ne possède aucun paramètre en sortie; le déroulement de son exécution est rapporté par un code de retour de valeur nulle (`ret_code = 0` ). Cela signifie que le pointeur du fichier a été déplacé, sinon, en cas d'erreur, sa position reste inchangée.

#### 6.3.5. La fonction `tp2_005_close`

-----

--- = f ( `tpopenid` )

La fonction `tp2_005_close` a pour but de fermer un fichier séquentiel.

Le seul paramètre en entrée est:

- `tpopenid`: une variable de type pointeur vers un Short dont la valeur spécifie un fichier. Elle ne possède pas de paramètre en sortie.

La réussite de son exécution est signalée par un code de retour de valeur nulle (`ret_code = 0` ). Dans ce cas,



le fichier spécifié en entrée est fermé sinon le répertoire reste inchangé.

#### 4.3.6. La fonction tp2\_006\_delete

-----  
tpdata\_length = f (tpopenid, tprequired\_data\_length )

La fonction tp2\_006\_delete a pour but d'effacer un certain nombre de bytes contenus dans un fichier séquentiel. L'analyse de ses paramètres est la suivante:

Les paramètres en entrée sont:

- tpopenid: une variable de type pointeur vers un Short dont la valeur identifie un fichier sur lequel se trouvent les bytes qui doivent être effacés ( Short \*tpopenid )

- tprequired\_data\_length: une variable de type pointeur vers un long dont la valeur spécifie le nombre de bytes qui ne doivent pas être effacés. Ces bytes se situent entre le début du fichier et la position actuelle du pointeur associé à ce fichier ( Long \*tprequired\_data\_length ).

Le paramètre en sortie est:

- tpdata\_length: une variable de type pointeur vers un long dont la valeur indique la nouvelle longueur en bytes du fichier après l'exécution sans erreur de la fonction ( ret\_code = 0 ).

Si l'exécution de cette fonction n'a pas réussi, la longueur du fichier reste inchangée.

Après la spécification externe de ces six fonctions, nous allons exposer le traitement qui est relatif à



chacune d'entre elles.

#### 6.4. La spécification interne des fonctions du TPDU

---

Dans la partie concernant la spécification interne, nous reprenons systématiquement les six fonctions déjà évoquées et pour chacune nous détaillons le traitement qui leur est associé.

Avant de procéder à l'analyse des traitements, il convient d'informer le lecteur de deux aspects qui sont constants pour chaque fonction envisagée.

Tout d'abord, la nomenclature des erreurs, pouvant survenir au cours de l'exécution d'une fonction, respecte la hiérarchie qui est la suivante:

- ret\_code
- err\_class
- err\_value.

Le code de retour de la fonction sera soit:

- NO\_ERR: s'il n'y a eu aucune erreur au cours de l'exécution de la fonction. Les paramètres en sortie sont alors définis et leurs valeurs sont conformes aux post-conditions formulées à la section 6.3.

- TEMP\_ERR: si l'erreur survenue est une erreur provisoire, c'est-à-dire, une erreur qui n'est pas directement relative au traitement du module TPDU mais à l'appel d'une fonction extérieure à ce module.

err\_class = SYS\_ERR et err\_value = TPCEFERR: en effet les seules fonctions appelées par ce module sont des fonctions offertes par la bibliothèque SINIX.



- CALL\_ERR: si l'erreur survenue est une erreur d'appel, c'est-à-dire, une erreur syntaxique ou sémantique dans un paramètre en entrée de la fonction ( cfr 6.3. ).

err\_class = PAR\_ERR et err\_value spécifie le paramètre en entrée dont la valeur est erronée ( ex: TPOPENID si l'identificateur du fichier ne correspond pas à un fichier ouvert et présent dans le répertoire ).

Pour ces deux dernières valeurs du code de retour, les valeurs des paramètres en sortie ne sont pas définies.

Ensuite pour chaque fonction, le déroulement séquentiel des instructions relatives au traitement, respecte la structure suivante:

- controle séquentiel portant sur les propriétés syntaxiques et sémantiques des paramètres en entrée. Dès qu'un controle détecte la présence d'une erreur, la fonction est avortée et l'erreur est mentionnée au moyen de la structure adéquate reprise ci-dessus ( CALL\_ERR, SYS\_ERR, ... )

- appel séquentiel des différentes fonctions de la bibliothèque SINIX permettant de réaliser le traitement demandé. Après l'exécution de chaque appel, un controle du code de retour de la fonction est effectué: si la valeur se révèle incorrecte, la fonction est avortée et l'erreur est mentionnée au moyen de la structure d'erreur adéquate présentée ci-dessus ( TEMP\_ERR, SYS\_ERR, TPCEFERR )

- mise à jour de l'élément de la liste: modification éventuelle des champs de l'élément concernant le fichier

sur lequel la fonction a été réalisée.

Après avoir analysé la structure générale des traitements, nous allons passer en revue chaque fonction du TPDU en relevant les différents appels à la bibliothèque SINIX qui ont été réalisés au sein de chacune d'elles.

#### 6.4.1. La fonction tp2\_001\_open

-----

Dans la fonction tp2\_001\_open, trois appels à la bibliothèque SINIX ont été effectués.

Ils concernent la fonction:

- open pour ouvrir le fichier
- stat pour connaître les informations sur le fichier ouvert dans le but de compléter l'élément de la liste créée pour ce fichier
- close pour fermer le fichier si la fonction stat renseigne une longueur négative pour le fichier.

#### 6.4.2. La fonction tp2\_002\_read

-----

Dans la fonction tp2\_002\_read, un appel à la bibliothèque SINIX a été effectué.

Il concerne la fonction:

- read pour lire à partir du fichier un nombre de bytes.



#### 6.4.3. La fonction tp2\_003\_write

---

Dans la fonction tp2\_003\_write, trois appels à la bibliothèque SINIX ont été effectués.

Ils concernent la fonction:

- write pour écrire un nombre de bytes sur le fichier
- fstat pour connaître les informations sur le fichier ouvert dans le but de mettre à jour l'élément de la liste relatif à ce fichier
- close pour fermer le fichier si la fonction fstat renseigne une longueur négative pour le fichier.

#### 6.4.4. La fonction tp2\_004\_position

---

Dans la fonction tp2\_004\_position, un appel à la bibliothèque SINIX a été effectué.

Il concerne la fonction:

- lseek pour déplacer le pointeur du fichier à un certain endroit à l'intérieur de celui-ci.

#### 6.4.5. La fonction tp2\_005\_close

---

Dans la fonction tp2\_005\_close, un appel à la bibliothèque SINIX a été effectué.

Il concerne la fonction:

- close pour fermer le fichier.

N.B. l'élément de la liste relatif au fichier qui est fermé, est enlevé de la liste.

#### 6.4.6. La fonction tp2\_006\_delete

-----

Dans la fonction tp2\_006\_delete, deux fonctions du module TPDU sont utilisées:

a) - tp4\_016\_delete\_all: pour vider complètement un fichier lorsque le nombre de bytes qui ne doivent pas être effacés est nul. Dans ce cas le fichier est fermé, retiré du répertoire et un nouveau fichier ( vide ) portant le même nom est créé. Les fonctions de la bibliothèque SINIX utilisées sont les suivantes:

- close pour fermer le fichier
- unlink pour enlever le fichier du répertoire
- creat pour créer un nouveau fichier dans le répertoire.

b) - tp4\_026\_delete\_part: pour effacer un nombre de bytes dans le fichier sans que celui-ci soit complètement vidé. Pour réaliser cette opération, un nouveau fichier ( fichier de travail ) est créé, les bytes qui ne doivent pas être effacés sont recopiés par paquets ( longueur maximale d'un paquet = 1024 bytes ). Le fichier source est ensuite fermé et enlevé du répertoire. Enfin, le nom du fichier de travail est changé et devient le nouveau fichier dans lequel sont stockés les bytes non effacés.

Pour réaliser ce traitement, les fonctions de la bibliothèque SINIX utilisées sont les suivantes:

- close pour fermer le fichier où sont stockés les bytes qui doivent être effacés ( fichier source )
- open pour récupérer ce fichier en lecture
- mktemp pour inventer un nom, non encore existant



dans le répertoire. Ce nom sera celui du fichier de travail.

- creat pour créer le fichier de travail dans le répertoire
- read pour lire les bytes dans le fichier source
- write pour écrire les bytes dans le fichier de travail
- close pour fermer le fichier source
- unlink pour enlever le fichier source du répertoire
- link pour attribuer le nom du fichier source au fichier de travail
- fstat pour connaître les informations sur le nouveau fichier source:
- close pour fermer le nouveau fichier source si sa longueur renseignée par fstat est négative.

Le listing du programme est fourni en annexe, le lecteur pourra facilement retrouver les différents traitements associés aux fonctions du TPDU.

La documentation complète du programme est insérée dans le listing conformément aux habitudes prises par SIEMENS.

#### 6.5. Le test du nouveau TPDU

-----

Pour réaliser le test du module avant de l'utiliser dans l'architecture, un programme appelant ce module a été réalisé.

Pour cette étape, nous avons pu réutiliser le

programme du test construit pour le controle de l'ancienne version du TPDU.

Le test se présente sous la forme d'un menu. L'utilisateur choisit une des six opérations traitées par le TPDU, entre des valeurs pour les paramètres en entrée et attend le résultat.

Le code de retour ainsi que les valeurs des paramètres en sortie lui sont transmis sur l'écran.

La difficulté d'un tel type de test réside dans le choix des valeurs critiques pour les paramètres en entrée. Il est parfois difficile de relever toutes les combinaisons et donc certains chemins d'exécution peuvent être oubliés.

#### 6.6. La conclusion du travail

-----

Pour conclure le chapitre relatif au travail, nous pouvons faire quelques remarques.

La technique de documentation standardisée qui est utilisée chez SIEMENS SOFTWARE nous a permis de comprendre facilement la spécification des programmes que nous avons eu l'occasion de consulter préalablement à la réalisation du travail.

Pour ce qui concerne l'organisation du travail, l'étape qui a requis le plus de temps est la spécification interne avec le choix des fonctions SINIX pour réaliser le traitement des opérations.

Le recours au langage de programmation " C ", nous a offert certains avantages telles que les définitions de



liste, l'utilisation de la bibliothèque SINIX.

Enfin, nous tenons à remercier pour l'aide apportée, les personnes de l'équipe au sein de laquelle nous avons travaillé.

## CONCLUSION GENERALE

-----

Ce mémoire nous a permis d'aborder la téléinformatique qui est de plus en plus sollicitée ces dernières années. Dans cette branche de l'informatique, le service de courrier électronique ou encore appelé messagerie électronique permet à des utilisateurs éloignés de s'échanger des messages.

Nous avons constaté qu'étant donné l'omniprésence de ce service dans tous les réseaux à travers le monde, il était nécessaire qu'un modèle détermine le standard que suivraient les constructeurs de logiciels. Les produits ainsi réalisés pourraient facilement s'interconnecter. Le CCITT publia en 1984 la série X400 spécifiant le modèle MHS, un standard pour le service de courrier électronique. Ce modèle, largement utilisé par les constructeurs, s'insère au sommet de l'architecture OSI.

Dans ce domaine, il existe d'autres modèles proposés tels que le MMFS utilisé dans le MAP pour les processus industriels.

Dans le cadre de ce mémoire, nous avons étudié le modèle MHS et plus particulièrement l'entité SDE avec son protocole de communication, le protocole P3.

Nous avons ensuite analysé le RQS qui permet de structurer le protocole P3 afin de réaliser les opérations à distance offertes par ce protocole.

Nous informons le lecteur que le RQS peut aussi être utilisé pour structurer d'autres protocoles tels que le



protocole P7 d'accès à la boîte aux lettres dans les stations de travail.

Comme nous pouvons le constater, le domaine dans lequel fut réalisé ce mémoire est très vaste et pourrait encore faire l'objet de recherches futures.

## BIBLIOGRAPHIE

-----

( Quart 86 ) J.S. Quarterman and J.C. Hoskins " Notable computer networks "

Communications of the A.C.M. Oct. 1986. Vol. 29, Num. 10 p.932 to 971.

( Mac 83 ) C.Macchi, J.F. Guilbert " Téléinformatique " Ed. Bordas Paris 1983

( Mah 84 ) R. Mahl et P. Berger " Les réseaux de la recherche " Technique et Science Informatique 1984, Vol. 3, Num. 10 p. 457 à 473

( Brun 86 ) J.Brun Notes du cours Télé-informatique et réseaux année académique 1985-1986

( Cun 83 ) I. Cunningham " Message Handling Systems and Protocols " Proceedings of the IEEE Dec. 1983, Vol. 71, Num. 12 p. 1425 TO 1430

( Mye 84 ) T.H. Myer " The MHS recommendations, structure and implementation perspective " IEEE/E Lsevier Science Publishers B.V. ( North Holland ) 1984

( Mal 86 ) S. Malde " Message Handling Standards " Computer communications Apr. 1986, Vol. 9, Num. 2



( Mau 86 ) T.I. Maude " Distribution lists in an  
X.400, Message Handling System " Computer communications  
Oct. 1986, Vol. 9, Num. 5

( Geh 85 ) N.Gehani " C: An Advanced Introduction "  
Bell Telephone Laboratories, Inc 1985, Printed in the  
U.S.A.

( Kel 84 ) A. Kelley " A book on C " The  
Benjamin/Cummings Publishing Company, Inc 1984

( X400 ) CCITT Recommendation X400 " Message Handling  
Systems: system model-service elements "  
Malaga-Torremolinos, 1984

( X409 ) CCITT Recommendation X409 " Message Handling  
Systems: presentation transfer syntax and notation "  
Malaga-Torremolinos, 1984

( X410 ) CCITT Recommendation X410 " Message Handling  
Systems: remote operations and reliable transfer server "

( X411 ) CCITT Recommendation X411 " Message Handling  
Systems: Message Transfer Layer " Malaga-Torremolinos,  
1984.





Annexe du mémoire

Vincent Laurent

Septembre 1987

```

1  /*daton *****/
2  /*
3  /* TYPE      : MODULE
4  /*
5  /* NAME      : open.c
6  /*
7  /* AUTOR     : V.Laurent      D AP11
8  /*
9  /* DATE      : 9.12.86
10 /*
11 /* COMPONENTE : TPDU
12 /*
13 /* DOC.-NR.   :
14 /*
15 /*
16 /* PRD#/VERS. :
17 /*
18 /* DESCRIPTION : look to the functionheader
19 /*
20 /*
21 /* SYSTEMABHAENGIGKEIT:
22 /*
23 /*
24 /* HISTORY :
25 /*
26 /* Vers.Nr. | Date | Changes | KI | CR# | FM# */
27 /* 1.0      | 9.12.86 | Original | V.L |
28 /*          |      |         |     |
29 /*datoff *****/
30
31
32
33 /*****/
34 /*
35 /*          I N C L U D E S
36 /*
37 /*****/
38
39          /***** Operating System - Includes *****/
40
41 #include <stdio.h>
42 #include <sys/types.h>
43 #include <sys/stat.h>
44 #include <sys/fcntl.h>
45
46          /***** EM-V1 general Includes *****/
47
48 #include <emsystem.h>
49
50          /***** Part system - extern *****/
51
52          /* Interface IMAI */
53
54 #include <malgcfe.h>
55 #include <magenex.h>
56
57          /***** Part system - intern *****/
58
59          /* TP - Global Definitions */
60 #include <tp2.h>
61
62          /* ENTRY- EXIT and ERROR-Makros */
63 #include <rt4log.h>
64
65          /* Listen */
66 #include <rt4list.h>
67 #include <tp4list.h>
68

```



```

72  /*                                                                    */
73  /*          D E F I N E S          */
74  /*                                                                    */
75  /*****
76
77
78          /***** Makros *****/
79
80          /***** Constants *****/
81
82          /* max. length of the text for the log-file*/
83  #define TPAR_LEN      90
84
85          /*****Error text*****/
86
87  #define TP2ERR001 " Error in determination of the filelength "
88
89
90  _EJECT_
91  /*****
92  /*                                                                    */
93  /*          D A T A          D E C L A R A T I O N          */
94  /*                                                                    */
95  /*****
96
97          /*IMPORT*/
98  /*  NO  */
99
100
101          /*EXPORT*/
102
103  EXPORT Rtlh      * tpanchor = (Rtlh *)NULL; /* Head of the TP-Liste */
104
105
106  _EJECT_
107  /*****
108  /*                                                                    */
109  /*          F U N C T I O N S          D E K L A R A T I O N          */
110  /*                                                                    */
111  /*****
112
113
114          /***** IMPORT *****/
115
116          /* functions for the list */
117  IMPORT Rtlh      * rt4_611_createlisthead ();
118  IMPORT struct Rtle * rt4_612_create_and_append ();
119  IMPORT Void      rt4_613_delete_any ();
120
121  IMPORT Short     ma2_003_log (); /* logging function */
122
123          /***** EXPORT *****/
124
125  EXPORT Short     tp2_001_open (); /* tp-function */
126
127
128
129  _EJECT_
130  /*exon *****/
131  /*                                                                    */
132  /* TYPE          : C-FUNCTION          */
133  /*                                                                    */
134  /* NAME          : tp2_001_open          */
135  /*                                                                    */
136  /* AUTHOR        : V.Laurent, D AP 11          */
137  /* DATE          : 9.12.86          */
138  /*                                                                    */
139  /*                                                                    */
140  /* SYNTAX        : Short tp2 001 open( oblk )          */

```

```

144 /*          Open a file for read or write          */
145 /*          */
146 /* INPUT PARAMETER          */
147 /*          */
148 /*          Pbhead  *pbhead          Standard Head          */
149 /*          Char    *tpaccess_method Access method of the file */
150 /*          Char    *tpdata_description Name of the file          */
151 /*          Tpdire  *tpdire          Mode in which the file          */
152 /*          has been opened.          */
153 /*          typedef enum (TPIN, TPOUT) TPDIRE Enumeration of the 2 modes */
154 /*          allowed:          */
155 /*          TPIN for READ          */
156 /*          TPOUT for WRITE          */
157 /* OUTPUT PARAMETER          */
158 /*          */
159 /*          Short    *tpopenid          File Identifier          */
160 /*          */
161 /*          */
162 /* RETURNVALUE          :          */
163 /*          */
164 /*          Short    ret_code = NO_ERR          */
165 /*          = TEMP_ERR          */
166 /*          = CALL_ERR          */
167 /*          */
168 /* STANDARD HEAD          :          */
169 /*          */
170 /*          Version = TP2VD1          */
171 /*          */
172 /*          ret_code = TEMP_ERR          */
173 /*          err_class = SYS_ERR          */
174 /*          err_value = TPCEFERR Error in the call of the c          */
175 /*          function          */
176 /*          ret_code = CALL_ERR          */
177 /*          err_class = PAR_ERR          */
178 /*          err_value = TPACCESS : Access method is          */
179 /*          not allowed          */
180 /*          = TPNODEFDIR : The direction of the          */
181 /*          exploitation is not defined */
182 /*          */
183 /*          err_class = ILL_VERS          */
184 /*          err_value = TPNOSUPVS : The version number is          */
185 /*          wrong.          */
186 /* GLOBAL DATA (produced) :          */
187 /*          */
188 /*          no          */
189 /*          */
190 /* GLOBAL DATA (changed) :          */
191 /*          */
192 /*          no          */
193 /*          */
194 /*          */
195 /* CALLS FUNCTIONS/PROGRAM/MAKROS:          */
196 /*          */
197 /*          open          */
198 /*          stat          */
199 /*          close          */
200 /*          */
201 /*          rt4_611_createlisthead          */
202 /*          rt4_612_create_and_append          */
203 /*          rt4_613_delete_any          */
204 /*          */
205 /*          CHECK_VERSION          */
206 /*          ENTRY          */
207 /*          EXIT          */
208 /*          ERROR          */
209 /*          */
210 /* REMARKS          :          */
211 /*          */
212 /*          For having at any time the file opened and the information

```



```

216 /*      a TPDU_identifier, the file name, the length of the file, his      */
217 /*      direction of exploitation, the data_length (the position          */
218 /*      since the beginning of the file pointer) and the offset          */
219 /*      (the length in bytes of the last movement of the file pointer) */
220 /*      After each operation on the file , the corresponding list        */
221 /*      element is updated. When the file is closed the list element      */
222 /*      is retrieved.                                                     */
223 /*      .                                                                    */
224 /*exoff ******/
225 _EJECT_
226
227 Short tp2_001_open( ptp1001 )
228 Ptp1001 *ptp1001;                                     /* Block of parameter */
229
230 Entr                                           /*@ TP_OPEN */
231
232 /*****
233 /*      Declaration
234 /*****
235
236 FASTAUTO Rtletp *hpointer;                      /* Pointer to the list element */
237 FASTAUTO Pbhead *tpbhead;                       /* TP-Standard Head */
238 FASTAUTO Short fdes;                            /* The return value of OPEN */
239 AUTO struct stat tpinf;                         /* The structure returned by FSTAT */
240
241                                           /* Useful definition for MA-Makros */
242 STATIC String fname = "tp2_001";
243 AUTO Char params[TPPAR_LEN];
244 STATIC Char errtext[] = TP2ERR001;
245 AUTO Pbhead class_value;
246
247 /*****
248 /*      Initialisation
249 /*****
250
251 Begin                                           /*@*/
252 tpbhead = ptp1001->pbhead;
253 tpbhead->ret_code = NO_ERR;
254
255 sprintf( params, " %d %s %s ", *ptp1001->tpdir,
256                                     ptp1001->tpdata_description,
257                                     ptp1001->tpaccess_method );
258 ENTRY (fname, params);
259
260 /*****
261 /*@      Check the INPUT parameter
262 /*****
263
264                                           /* Check the version */
265 If
266     ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tpbhead->version))
267 Then
268
269     tpbhead->ret_code = CALL_ERR;
270     tpbhead->err_class = ILL_VERS;
271     tpbhead->err_value = TPNOSUPVS;
272     EXIT ( fname, tpbhead->ret_code );
273     return ( tpbhead->ret_code );
274 Fi
275
276                                           /* Check the access mode */
276 If
277     ( strcmp(ptp1001->tpaccess_method, "TPSAM" ) != 0 )
278 Then
279     tpbhead->ret_code = CALL_ERR;
280     tpbhead->err_class = PAR_ERR;
281     tpbhead->err_value = TPACCESS;
282     EXIT ( fname, tpbhead->ret_code );
283     return ( tpbhead->ret_code );
284 Fi

```



```

288      (
289      (*(ptp1001->tpdir) != TPIN)                                /* read */
290      &&
291      (*(ptp1001->tpdir) != TPOUT)                               /* write */
292      )
293  Then
294      tppbhead->ret_code = CALL_ERR;
295      tppbhead->err_class = PAR_ERR;
296      tppbhead->err_value = TPNODEFDIR;
297      EXIT ( fname, tppbhead->ret_code );
298      return ( tppbhead->ret_code );
299  Fi
300
301
302  If                                     /*& There is no list head */
303      ( tpanchor == (Rtlh *) NULL )
304  Then                                     /*&*/
305      /*****
306      /*&      Creat the head of the list element          */
307      /*****
308      tpanchor = (Rtlh *)rt4_611_createlisthead ( sizeof (Rtlh) );
309  Fi                                     /*&*/
310
311      /*****
312      /*&      Creat an listelement corresponding to the file      */
313      /*****
314
315      hpointer =
316      (Rtlatp *)rt4_612_create_and_append((Rtlh *)tpanch, sizeof(Rtlatp));
317
318      /*****
319      /*&      call OPEN                                          */
320      /*****
321
322
323  If ( ( *ptp1001->tpdir ) == TPIN )
324
325      Then                                     /* read */
326      fdes = open(ptp1001->tpdata_description,O_RDONLY);
327      Else                                     /* write */
328      fdes = open(ptp1001->tpdata_description,O_WRONLY);
329  Fi
330
331
332  If                                     /*& Check the return value of OPEN */
333      ( ( fdes ) < 0 )
334  Then                                     /*& Error in call of OPEN */
335                                          /*& Delete the list element */
336                                          /*& ret_code = ERROR */
337      rt4_615_delete_any (tpanch, (struct Rtla *)hpointer);
338      tppbhead->ret_code = TEMP_ERR;
339      tppbhead->err_class = SYS_ERR;
340      tppbhead->err_value = TPCEFERR;
341  Else                                     /*&*/
342
343      /*****
344      /*&      Determine the file length                      */
345      /*****
346
347      If                                     /*& Check the return value of STAT */
348      ( ( stat(ptp1001->tpdata_description,&tpinf) ) != 0 )
349      Then                                     /*& Error in call of STAT */
350                                          /*& Return_code = ERROR */
351      tppbhead->ret_code = CALL_ERR;
352      tppbhead->err_class = PAR_ERR;
353      tppbhead->err_value = TPOPENID;
354
355      Else                                     /*& Test the value of the length of the file */
356      If                                     /*& The length is / 0 */

```



```

360                                     /* Close the file */
361                                     /* Close the file */
362                                     tppbhead->ret_code = TEMP_ERR;
363                                     tppbhead->err_class = SYS_ERR;
364                                     tppbhead->err_value = TPCFERR;
365                                     close(fdes);
366
367                                     /* Delete the list element */
368                                     rt4_615_delete_any ( tpanchor, (struct Rtle *)hpointer);
369
370                                     /* MA-ERROR call */
371                                     class_value.err_class = SYS_ERR;
372                                     class_value.err_value = TPNFLERR;
373                                     ERROR ( fname, TP2ERR001 , class_value );
374
375                                     Else /* Put the length into the list element */
376                                     hpointer->rtlebtp.filalength = (Long)tpinf.st_size;
377                                     Fi                                     /* */
378                                     /* */
379
380                                     /* Fill in the rest of the list element fields */
381                                     hpointer->rtlebtp.filename = ptp1001->tpdata_description;
382                                     hpointer->rtlebtp.tpfileid = fdes;
383                                     hpointer->rtlebtp.dvfileid = fdes;
384                                     hpointer->rtlebtp.modus = *(ptp1001->tpdir);
385                                     hpointer->rtlebtp.offset = QL;
386                                     hpointer->rtlebtp.datalength = QL;
387                                     /* Position the file pointer at the beginning of the file */
388
389                                     /* Complete the OUTPUT parameter */
390                                     *(ptp1001->tpopenid) = fdes;
391                                     Fi                                     /* */
392                                     /* return (ret_code) */
393                                     EXIT ( fname, tppbhead->ret_code );
394                                     return ( tppbhead->ret_code );
395
396                                     Send                                     /* */
397                                     End                                     /* */

```

```

1  /*data ******/
2  /* */
3  /* TYPE      : MODULE */
4  /* */
5  /* NAME      : read */
6  /* */
7  /* AUTHOR    : V.Laurent      D AP11 */
8  /* */
9  /* DATE      : 18.12.86 */
10 /* */
11 /* COMPONENTE : TPDU */
12 /* */
13 /* DOC.-NR.   : */
14 /* */
15 /* */
16 /* PRD#/VERS. : */
17 /* */
18 /* DESCRIPTION : */
19 /* */
20 /* */
21 /* SYSTEMASHAENGISKEIT: */
22 /* */
23 /* */
24 /* HISTORY   : */
25 /* */
26 /* Vers.Nr. | Date | Changes | KZ | CR# | FM# */
27 /* 1.0      | 18.12.86 | Original | V.LI */
28 /*          |          |          |      |      |      */
29 /*dataoff ******/
30
31
32
33 /*****
34 /*
35 /*          I N C L U D E S
36 /*
37 /*****
38
39          /***** Operating System - Includes *****/
40
41
42 #include <stdio.h>
43 #include <sys/types.h>
44 #include <sys/stat.h>
45
46
47          /***** EM-V1 general Includes *****/
48
49 #include <emsystem.h>
50
51
52
53          /*****Part system - extern *****/
54
55
56          /* Interface IMAI */
57 #include <malgcfe.h>
58 #include <magenex.h>
59
60
61
62
63          /*****Partr system - intern *****/
64
65          /* TP - Global Definitions */
66 #include <tp2.h>
67
68          /* ENTRY- EXIT and ERROR-M-... */

```



```

72 #include <rt4list.h>
73 #include <tp4list.h>
74
75
76
77
78 _EJECT_
79 /*****
80 /*
81 /*          D E F I N E S
82 /*
83 /*****
84
85
86          /***** Makros *****/
87
88          /***** Constants*****/
89
90          /* max. Laength of a Text for the Login file */
91 #define TPAR_LEN      90
92 #define RTLN_LOGTEXT  60
93 #define TEST_LOGTEXT 150
94
95          /*****Error text*****/
96
97 #define TPERRORO " Error in f_length"
98
99
100 _EJECT_
101 /*****
102 /*
103 /*          D A T A          D E C L A R A T I O N
104 /*
105 /*****
106
107          /***** IMPORT *****/
108
109          /* Head of the P-Lists */
110 IMPORT Rtlh      * tpanchor;
111
112
113          /***** EXPORT *****/
114
115 /* NO      */
116
117
118
119 _EJECT_
120 /*****
121 /*
122 /*          F U N C T I O N          D E C L A R A T I O N
123 /*
124 /*****
125
126
127          /***** IMPORT *****/
128
129          /* Functions for the list */
130 IMPORT struct Rtle * rt4_614_search ();
131 IMPORT Short      tp4_001_search_tpopenid ();
132
133
134          /* Function for the login */
135 IMPORT Short      mz2_003_log ();
136
137
138          /***** EXPORT *****/
139
140

```

```

144
145 _EJECT_
146 /**exon *****/
147 /* */
148 /* TYPE : C-FUNKTION */
149 /* */
150 /* NAME : tp2_002_read */
151 /* */
152 /* AUTOR : V.Laurent, D AP 11 */
153 /* DATE : 18.12.86 */
154 /* */
155 /* SYNTAX : Short tp2_002_read( pblk ) */
156 /* */
157 /* */
158 /* DESCRIPTION : */
159 /* */
160 /* Read sequentially a certain number of bytes from a file */
161 /* */
162 /* */
163 /* INPUT PARAMETER : */
164 /* */
165 /* Pthead *pthead Standard Head */
166 /* Short *tpopenid File identifier */
167 /* Char *tpinbufferaddress Address of a buffer */
168 /* Short *tpinrequired_data_length Number of bytes */
169 /* required to read */
170 /* */
171 /* */
172 /* OUTPUT PARAMETER : */
173 /* */
174 /* Short * tpread_data_length Number of bytes readen */
175 /* Bool * tpinfurther_data Is there still to read in the */
176 /* file? YES NO */
177 /* */
178 /* RETURNVALUE : */
179 /* */
180 /* Short ret_code = NO_ERR */
181 /* = TEMP_ERR */
182 /* = CALL_ERR */
183 /* */
184 /* STANDARD HEAD : */
185 /* */
186 /* Version = TP2V01 */
187 /* */
188 /* ret_code = TEMP_ERR */
189 /* err_class = SYS_ERR */
190 /* err_value = TPOCFERR Error in the execution of the */
191 /* c function */
192 /* ret_code = CALL_ERR */
193 /* err_class = PAR_ERR */
194 /* err_value = TPOPNID : Unknown file identifier */
195 /* = TPINBUFFER : Wrong address of the buffer */
196 /* = TPINREQUIRED : Wrong number of bytes to */
197 /* read */
198 /* */
199 /* err_class = ILL_VERS */
200 /* err_value = TPNOSUPVS :not allowed number of version */
201 /* */
202 /* GLOBAL DATA (produced) : */
203 /* */
204 /* no */
205 /* */
206 /* */
207 /* GLOBAL DATA ( changed); */
208 /* */
209 /* no */
210 /* */
211 /* */
212 /* CALLS FUNCTIONS/PROGRAM/MACROS. */

```



```

216 /*
217 /*
218 /*          rt4_614_search
219 /*
220 /*          CHECK_VERSION
221 /*          ENTRY
222 /*          EXIT
223 /*
224 /*
225 /* REMARKS :
226 /*      When the file is readed with succes the file pointer is
227 /*      moved of a length equal to the number of bytes readed or
228 /*      is positionned at the EOF if EOF is reached during the lecture*/
229 /*
230 /*eof *****/
231 _EJECT_
232
233
234 Short tp2_002_read( ptp1002 )
235 Ptp1002 *ptp1002;          /*Block of parameter */
236
237 Entr          /*& TP_READ */
238
239 /*****/
240 /*          Declaration
241 /*****/
242
243 FASTAUTO Rtletp *hpointer;          /* Pointer to the list element */
244 FASTAUTO int rdlgth;          /* Return value of READ */
245 FASTAUTO Pbhead *tpbhead;          /* TP-Standard Headr*/
246
247          /*Useful definitions for MA-Makros */
248
249 STATIC String fname = "tp2_002";
250 AUTO Char params[TPPAR_LEN];
251
252 /*****/
253 /*          Initialisation
254 /*****/
255
256 Begin          /*&*/
257
258 tpbhead = ptp1002->pbhead;
259 tpbhead->ret_code = NO_ERR;
260
261          /*& MA_LOG - Call*/
262 sprintf(params, "%d %d",
263          *ptp1002->tpopenid, *ptp1002->tpinrequired_data_length);
264 ENTRY (fname, params);
265
266 /*****/
267 /*&          Check the INPUT parameters
268 /*****/
269
270 If          /* Check the version number */
271      ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tpbhead->version))
272 Then
273
274      tpbhead->ret_code = CALL_ERR;
275      tpbhead->err_class = ILL_VERS;
276      tpbhead->err_value = TPNOSUPVS;
277      EXIT ( fname, tpbhead->ret_code );
278      return ( tpbhead->ret_code );
279 Fi
280
281          /* Search the list element in the list */
282 hpointer = (Rtletp *)rt4_614_search( tpanchor, tp4_001_search_tpopenid,
283          ptp1002->tpopenid );
284

```

```

288     tppbhead->ret_code = CALL_ERR;
289     tppbhead->err_class = PAR_ERR;
290     tppbhead->err_value = TPOPENID;
291     EXIT ( fname, tppbhead->ret_code );
292     return ( tppbhead->ret_code );
293 Fi
294
295 If                                     /* Check if the file is opened to read */
296     ( hpointer->rtlebt.modus != TPIN )
297 Then
298     tppbhead->ret_code = CALL_ERR;
299     tppbhead->err_class = PAR_ERR;
300     tppbhead->err_value = TPDIRUSE;
301     EXIT ( fname, tppbhead->ret_code );
302     return ( tppbhead->ret_code );
303 Fi
304
305 If                                     /* Check if the number of bytes to read is not < 0 */
306     ( *ptp1002->tpinrequired_data_length < 0 )
307 Then
308     tppbhead->ret_code = CALL_ERR;
309     tppbhead->err_class = PAR_ERR;
310     tppbhead->err_value = TPINREQUIRED;
311     EXIT ( fname, tppbhead->ret_code );
312     return ( tppbhead->ret_code );
313 Fi
314
315 If                                     /* Check if the address of the buffer is correct */
316     ( ptp1002->tpinbufferaddress == NULL )
317 Then
318     tppbhead->ret_code = CALL_ERR;
319     tppbhead->err_class = PAR_ERR;
320     tppbhead->err_value = TPINBUFFER;
321 #ifdef HOWFAR
322     EXIT ( fname, tppbhead->ret_code );
323 #endif
324     return ( tppbhead->ret_code );
325 Fi
326
327 /*****
328 /*@      The file to read contains 0 byte      */
329 /*****/
330
331 If
332     ( hpointer->rtlebt.filelength == 0 )
333 Then
334     *ptp1002->tpread_data_length = 0;
335     *ptp1002->tpinfurther_data = FALSE;
336 #ifdef HOWFAR
337     EXIT ( fname, tppbhead->ret_code );
338 #endif
339     return ( tppbhead->ret_code );
340 Fi
341
342 /*****
343 /*      The number of byte to read is 0      */
344 /*****/
345
346 If
347     ( *ptp1002->tpinrequired_data_length == 0 )
348 Then
349     *ptp1002->tpread_data_length = 0;
350     *ptp1002->tpinfurther_data = YES;
351 #ifdef HOWFAR
352     EXIT ( fname, NO_ERR );
353 #endif
354     return ( NO_ERR );
355 Fi
356
357

```



```

360  /*****
361
362
363
364  rdlength = ( read( hpointer->rtleabtp.dvfileid,ptp1002->tpinbufferaddress,
365                  *ptp1002->tpinrequired_data_length));
366
367
368
369  If                                     /*& Ret_code = NO_ERROR  */
370      ( ( rdlength ) > 0 )
371  Then                                     /*&*/
372                                          /*& No error in the READ call */
373  /*****
374  /*&      Position the file pointer in the list element      */
375  /*****
376
377      hpointer->rtleabtp.offset = 0L;
378      hpointer->rtleabtp.datalength += (Long) (*ptp1002->tpinrequired_data_length);
379      /*& Determine if we are at the EOF  */
380      *ptp1002->tpinfurther_data =
381      ( hpointer->rtleabtp.datalength < hpointer->rtleabtp.filelength );
382
383      /*& Complete the output parameter */
384      *ptp1002->tpread_data_length = rdlength ;
385
386  Else                                     /*&*/
387
388      If                                     /*& Error in the execution of read */
389          ( ( rdlength ) == -1 )
390      Then                                     /*& Ret_Code = ERROR  */
391          tppbhead->ret_code = TEMP_ERR;
392          tppbhead->err_class = SYS_ERR;
393          tppbhead->err_value = TPCEFERR;
394
395      Else                                     /*& The EOF is reached  */
396          hpointer->rtleabtp.offset = 0L;
397          hpointer->rtleabtp.datalength = hpointer->rtleabtp.filelength;
398          /*& Complete the Output Parameters(no error)  */
399          *ptp1002->tpinfurther_data = NO;
400          *ptp1002->tpread_data_length = rdlength ;
401          Fi                                     /*&*/
402      Fi                                     /*&*/
403
404                                          /*& return ( ret_code ) */
405  EXIT ( fname, tppbhead->ret_code );
406  return ( tppbhead->ret_code );
407  Bend                                     /*&*/
408  End                                     /*&*/

```

```

1 /*daton *****/
2 /* */
3 /* TYPE : MODULE */
4 /* */
5 /* NAME : write */
6 /* */
7 /* AUTOR : V.Laurent D AP11 */
8 /* */
9 /* DATE : 18.12.86 */
10 /* */
11 /* COMPONENTE : TPDU */
12 /* */
13 /* DOC.-NR. : */
14 /* */
15 /* */
16 /* PRD#/VERS. : */
17 /* */
18 /* DESCRIPTION : */
19 /* */
20 /* */
21 /* SYSTEMABHAENGIGKEIT: */
22 /* */
23 /* */
24 /* HISTORY : */
25 /* */
26 /* Vers.Nr. | Date | Changes | KZ | CR# | FM# */
27 /* 1.0 | 18.12.86 | Original | V.L. | */
28 /* | | | | | */
29 /*datoff *****/
30
31
32
33 /* *****/
34 /* */
35 /* INCLUDES */
36 /* */
37 /* *****/
38
39 /****** Operating System - Includes *****/
40
41 #include <stdio.h>
42 #include <sys/types.h>
43 #include <sys/stat.h>
44
45
46 /****** EM-V1 general Includes *****/
47
48 #include <emsystem.h>
49
50
51
52 /*****Part system - extern *****/
53
54
55 /* Interface IMAI */
56 #include <malgcfe.h>
57 #include <magenex.h>
58
59
60
61
62 /****** Teilsystem - intern *****/
63
64 /* TP - Global Definitions */
65 #include <tp2.h>
66
67 /* ENTRY- EXITa nd RROR-Makros */
68 #include <tp2.h>

```



```

72 #include <tp4list.h>
73
74
75
76
77 _EJECT_
78 /*****
79  */
80  /*          D E F I N E S          */
81  */
82 /*****
83
84
85          ***** Makros *****
86
87          ***** Constants *****
88
89          /* max. Laength of a Text for the login file */
90 #define TPPAR_LEN      90
91
92          *****Error text*****
93
94 #define TP2ERR00 " Error in f_length"
95
96
97 _EJECT_
98 /*****
99  */
100 /*          D A T A          D E C L A R A T I O N          */
101  */
102 /*****
103
104          ***** IMPORT *****
105
106          /* Head of TP-Lists */
107 IMPORT Rtlh      * tpanchor;
108
109          ***** EXPORT *****
110
111 /*      NO      */
112
113
114
115 _EJECT_
116 /*****
117  */
118 /*          F U N C T I O N S          D E K L A R A T I O N          */
119  */
120 /*****
121
122
123          ***** IMPORT *****
124
125          /*      Functions for the list      */
126 IMPORT struct Rtlh * rt4_614_search ();
127 IMPORT Short      tp4_001_search_topenid ();
128
129          /*      Login function      */
130 IMPORT Short      ma2_003_log ();
131
132          ***** EXPORT *****
133
134          /*      tp-function      */
135 EXPORT Short tp2_003_write ();
136
137
138 _EJECT_
139 /*exon *****
140  */

```

```

144 /* */
145 /* AUTOR : V.Laurent, D AP 11 */
146 /* DATE : 18.12.86 */
147 /* */
148 /* SYNTAX : Short tp2_003_write( pblk ) */
149 /* */
150 /* */
151 /* DESCRIPTION : */
152 /* */
153 /* Write bytes in a file */
154 /* */
155 /* INPUT PARAMETER : */
156 /* */
157 /* Pbhead *pbhead Standard Head */
158 /* Short *tpopenid File Identifier */
159 /* Char *tpoutbufferaddress Address of a buffer */
160 /* Short *tpoutrequired_data_length The number of bytes */
161 /* required to write */
162 /* Bool *tpsecure Not yet defined */
163 /* */
164 /* */
165 /* OUTPUT PARAMETER : */
166 /* */
167 /* Short *tpwritten_data_length The number of bytes which */
168 /* has been really written. */
169 /* Bool *tpoutfurtherdataspace Is there still place to */
170 /* write in this file? */
171 /* YES NO */
172 /* */
173 /* RETURNVALUE : */
174 /* */
175 /* Short ret_code = NO_ERR */
176 /* = TEMP_ERR */
177 /* = CALL_ERR */
178 /* */
179 /* STANDARD HEAD : */
180 /* */
181 /* Version = TP2V01 */
182 /* */
183 /* ret_code = TEMP_ERR */
184 /* err_class = SYS_ERR */
185 /* err_value = TPCEFERR Error in the execution of the */
186 /* c function */
187 /* ret_code = CALL_ERR */
188 /* err_class = PAR_ERR */
189 /* err_value = TPOPENID :Unknown File Identifier */
190 /* = TPOUTREQUIRED : Wrong number of bytes to write*/
191 /* = TPOUTBUFFER : Wrong address of the buffer*/
192 /* = TPSECURE : Wrong value of the parameter*/
193 /* */
194 /* */
195 /* err_class = ILL_VERS */
196 /* err_value = TPNOSUPVS :Not allowed number of version */
197 /* */
198 /* */
199 /* GLOBAL DATA (produced) ; */
200 /* */
201 /* no */
202 /* */
203 /* GLOBAL DATA (changed) ; */
204 /* */
205 /* no */
206 /* */
207 /* */
208 /* CALLS FUNCTIONS/PROGRAM/MAKROS: */
209 /* */
210 /* close */
211 /* fstat */
212 /* write

```



```

216 /*          CHECK_VERSION          */
217 /*          ENTRY                    */
218 /*          EXIT                     */
219 /*          ERROR                    */
220 /*                                     */
221 /*                                     */
222 /* REMARKS      :                    */
223 /*                                     */
224 /*                                     */
225 /*exoff *****/
226 _EJECT_
227
228
229 Short tp2_003_write( ptp1003 )
230 Ptp1003 *ptp1003;                /* Block of parameter */
231
232 Entr                                /*& TP_WRITE */
233
234 /******
235 /*          Declaration          */
236 /******
237
238 FASTAUTO Rtletp *hpointer;        /* Pointer to list element */
239 FASTAUTO Pbhead *tpbhead;        /* TP-Standard Head */
240 FASTAUTO struct stat tpinf;      /* The structure returned by FSTAT */
241 FASTAUTO Long    oldfilelength;  /* The old file length */
242 FASTAUTO Long    newfilelength;  /* The new file length */
243
244 /* Useful definitions for MA-Makros */
245
246 STATIC String fname = "tp2_003";
247 AUTO Char    params[TPPAR_LEN];
248 STATIC String errtext = TP2ERR00;
249 AUTO Pbhead class_value;
250
251 /******
252 /*          Initialisation      */
253 /******
254
255 Begin                                /*&*/
256
257 tpbhead = ptp1003->pbhead;
258 tpbhead->ret_code = NO_ERR;
259
260 /*& MA_LOG -call */
261 sprintf( params, "%d %d %d", *ptp1003->tpopenid,
262          *ptp1003->tpoutrequired_data_length,
263          *ptp1003->tpsecure1;
264 ENTRY ( fname, params );
265
266 /******
267 /*&          Test the INPUT parameters          */
268 /******
269 /* Test the version number */
270
271 If
272     ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tpbhead->version))
273 Then
274
275     tpbhead->ret_code = CALL_ERR;
276     tpbhead->err_class = ILL_VERS;
277     tpbhead->err_value = TPNOSUPVS;
278     EXIT ( fname, tpbhead->ret_code );
279     return ( tpbhead->ret_code );
280 Fi
281
282 /* Search the list element in the list */
283 hpointer = (Rtletp *)rt4_614_search( tpanchor, tp4_001_search_tpopenid,
284                                     ptp1003->tpopenid );

```



```

288         tppbhead->ret_code = CALL_ERR;
289         tppbhead->err_class = PAR_ERR;
290         tppbhead->err_value = TPOPENID;
291 #ifdef HOWFAR
292     EXIT ( fname, tppbhead->ret_code );
293 #endif
294     return ( tppbhead->ret_code );
295 Fi
296
297 If                                     /* Check of the file is opened for writing */
298     ( hpointer->rtlbtpp.modus != TPOUT )
299     Then                               /* It is opened for reading */
300         tppbhead->ret_code = CALL_ERR;
301         tppbhead->err_class = PAR_ERR;
302         tppbhead->err_value = TPDIRUSE;
303 #ifdef HOWFAR
304     EXIT ( fname, tppbhead->ret_code );
305 #endif
306     return ( tppbhead->ret_code );
307 Fi
308
309 I f                                     /* Check the value of the number of bytes to WRITE */
310     ( *ptp1003->tpoutrequired_data_length < 0 )
311     Then                               /* The value is < 0 */
312         tppbhead->ret_code = CALL_ERR;
313         tppbhead->err_class = PAR_ERR;
314         tppbhead->err_value = TPOUTREQUIRED;
315 #ifdef HOWFAR
316     EXIT ( fname, tppbhead->ret_code );
317 #endif
318     return ( tppbhead->ret_code );
319 Fi
320
321 If                                     /* Check the buffer address */
322     ( ptp1003->tpoutbufferaddress == NULL )
323     Then                               /* Wrong buffer address */
324         tppbhead->ret_code = CALL_ERR;
325         tppbhead->err_class = PAR_ERR;
326         tppbhead->err_value = TPOUTBUFFER;
327 #ifdef HOWFAR
328     EXIT ( fname, tppbhead->ret_code );
329 #endif
330     return ( tppbhead->ret_code );
331 Fi
332
333                                     /* For the moment this parameter is */
334                                     /* not used , its value is always */
335                                     /* assigned to "YES" and it is */
336                                     /* considered as a constant. */
337 If
338     ( *ptp1003->tpsecure != YES )
339     Then
340         tppbhead->ret_code = CALL_ERR;
341         tppbhead->err_class = PAR_ERR;
342         tppbhead->err_value = TPSECURE;
343 #ifdef HOWFAR
344     EXIT ( fname, tppbhead->ret_code );
345 #endif
346     return ( tppbhead->ret_code );
347 Fi
348
349 /*****
350 /*@          Zed byte have been to write          */
351 /*****
352
353 If
354     ( *ptp1003->tpoutrequired_data_length == 0 )
355     Then
356         *ptp1003->tpoutbufferaddress = TRUE;

```



```

360 #endif
361     return ( NO_ERR );
362 Fi
363
364
365
366                                     /* Save the old file length */
367 oldfilelength = hpointer->rtlebbtp.filelength;
368                                     /* WRITE */
369
370
371
372 /*****
373 /*@          Call    WRITE          */
374 /*****/
375
376
377 If                                     /*@ Noerror */
378 ( ( write(hpointer->rtlebbtp.dvfileid,ptp1003->tpoutbufferaddress,
379          *ptp1003->tpoutrequired_data_length) ) > 0 )
380 Then                                     /*@*/
381
382                                     /*@ Determine the new file length */
383
384
385 /*****
386 /*          Call    FSTAT          */
387 /*****/
388     If                                     /*@ No error in FSTAT call */
389     ( ( fstat(hpointer->rtlebbtp.dvfileid,&tpinf) ) == 0 )
390     Then                                     /*@ Test if the length is > 0 */
391         newfilelength = (Long)tpinf.st_size;
392     If                                     /*@ Error : length is < 0 */
393     ( newfilelength < 0 )
394     Then                                     /*@ Close the file */
395
396         close(*ptp1003->tpopenid);
397                                     /*@ Delete the list element */
398         rt4_615_delete_any (tpanchor, (struct Rtle *)hpointer );
399
400                                     /* MA_ERROR */
401         class_value.err_class = SYS_ERR;
402         class_value.err_value = TPNFLERR;
403         ERROR ( fname, TP2ERR00, class_value);
404
405     Else                                     /*@ Update the list element */
406         If ( newfilelength > oldfilelength )
407         Then
408             hpointer->rtlebbtp.filelength = newfilelength;
409             Fi
410             hpointer->rtlebbtp.offset = 01;
411             *ptp1003->tpoutfurtherdataspace = YES;
412             hpointer->rtlebbtp.datalength +=
413                 *ptp1003->tpwritten_data_length ;
414                 /*@ Complete the OUTPUT parameter*/
415             *ptp1003->tpwritten_data_length =
416                 *ptp1003->tpoutrequired_data_length ;
417         Fi                                     /*@*/
418
419
420
421     Else                                     /*@ Ret_code = ERROR */
422         /* Error in the call of the function FSTAT*/
423         tppbhead->ret_code = TEMP_ERR;
424         tppbhead->err_class = SYS_ERR;
425         tppbhead->err_value = TPCEFERR;
426     Fi                                     /*@*/
427
428 Else

```

```
432 Fi                                     /*2*/
433                                     /*3 return (ret_code) */
434 EXIT ( fname, tppbhead->ret_code );
435 return ( tppbhead->ret_code );
436 Bend                                     /*2*/
437 End                                     /*3*/
```



```

1  /*daton *****/
2  /* */
3  /* TYPE      : MODULE */
4  /* */
5  /* NAME      : position.c */
6  /* */
7  /* AUTHOR    : V.Laurent      D AP11 */
8  /* */
9  /* DATE      : 15.12.86 */
10 /* */
11 /* COMPONENTE : TPDU */
12 /* */
13 /* DOC.-NR.   : */
14 /* */
15 /* */
16 /* PRD#/VERS. : */
17 /* */
18 /* DESCRIPTION : */
19 /* */
20 /* */
21 /* SYSTEMAPHAENGISKEIT: */
22 /* */
23 /* */
24 /* HISTORY   : */
25 /* */
26 /* Vers.Nr. | Date | Changes | KZ | CR# | FM# */
27 /* 1.0      | 15.12.86 | Original | V.L |      |      */
28 /*          |      |          |      |      |      */
29 /*datoff *****/
30
31
32
33 /*****/
34 /* */
35 /*          I N C L U D E S          */
36 /* */
37 /*****/
38
39          /***** Operating System - Includes *****/
40
41
42 #include <stdio.h>
43 #include <sys/types.h>
44 #include <sys/stat.h>
45
46
47          /***** EM-V1 general Includes *****/
48
49 #include <emsystem.h>
50
51
52
53          /*****Part lsystem - extern *****/
54
55
56          /*Interface IMAI */
57 #include <malgcf.h>
58 #include <magenex.h>
59
60
61
62          /***** Part system - intern *****/
63
64
65          /* TP - Global Definitions*/
66 #include <tp2.h>
67
68          /* ENTRY- EXIT and ERROR-Messages */

```

```

72 #include <rt4list.h>
73 #include <tp4list.h>
74
75
76
77
78 _EJECT_
79 /*****
80 /*
81 /*          D E F I N E S
82 /*
83 /*****/
84
85
86          /***** Makros *****/
87
88          /***** Constants *****/
89
90          /* max. Laength of a Text for the Login file*/
91 #define TPAR_LEN      90
92 #define RTLN_LOGTEXT  60
93 #define TEST_LOGTEXT 150
94
95          /*****Error text*****/
96
97 #define TPERROD " Error in thedetermination of the file length "
98
99
100 _EJECT_
101 /*****
102 /*
103 /*          D A T A          D E C L A R A T I O N
104 /*
105 /*****/
106
107          /***** IMPORT *****/
108
109          /* Head ofTP-Liste */
110 IMPORT Rtlh      * tpanchor;
111
112
113
114          /***** EXPORT *****/
115
116 /* NO */
117
118 _EJECT_
119 /*****
120 /*
121 /*          F U N C T I O N          D E C L A R A T I O N
122 /*
123 /*****/
124
125
126          /***** IMPORT *****/
127
128          /* Functions for the list */
129 IMPORT struct Rtlh * rt4_614_search ();
130 IMPORT Short      tp4_001_search_topenid ();
131
132
133          /* Login Function */
134 IMPORT Short      ma2_003_log ();
135
136
137          /***** EXPORT *****/
138
139          /* tp-Function */
140 EXPORT Short      ma2_003_log ();

```



```

144
145
146
147 _EJECT_
148 /*exon *****/
149 /* */
150 /* TYPE : C-FUNKTION */
151 /* */
152 /* NAME : tp2_004_position */
153 /* */
154 /* AUTOR V.Laurent, D AP 11 */
155 /* DATE : 15.12.86 */
156 /* */
157 /* SYNTAX : Short tp2_004_position( pblk ) */
158 /* */
159 /* */
160 /* DESCRIPTION : */
161 /* */
162 /* Position the file pointer */
163 /* */
164 /* INPUT PARAMETER : */
165 /* */
166 /* Pbhead *pbhead Standard Head */
167 /* Short *tpopenid File identifier */
168 /* Long *tpoffset The distance in bytes since the */
169 /* beginning of the file of the new */
170 /* position of the file pointer. */
171 /* */
172 /* */
173 /* */
174 /* OUTPUT PARAMETER : */
175 /* */
176 /* no */
177 /* */
178 /* RETURNVALUE : */
179 /* */
180 /* Short ret_code = NO_ERR */
181 /* = TEMP_ERR */
182 /* = CALL_ERR */
183 /* */
184 /* STANDARD HEAD : */
185 /* */
186 /* Version = TP2V01 */
187 /* */
188 /* ret_code = CALL_ERR */
189 /* err_class = PAR_ERR */
190 /* err_value = TPOPENID : Unknown openid */
191 /* = TPOFFSET : wrong Offset */
192 /* */
193 /* err_class = ILL_VERS */
194 /* err_value = TPNOSUPVS : Mot allowed number of version*/
195 /* */
196 /* */
197 /* GLOBAL DATA (produced); */
198 /* */
199 /* no */
200 /* */
201 /* GLOBAL DATA (changed); */
202 /* */
203 /* no */
204 /* */
205 /* */
206 /* CALLS FUNCTIONS/PROGRAM/MAKROS: */
207 /* */
208 /* lseek */
209 /* */
210 /* */
211 /* rt4_614_search */
212 /*

```

```

216 /*
217 /*
218 /* REMARKS :
219 /*
220 /*
221 /*incff *****/
222
223
224
225
226
227 Short tp2_004_position( ptp1004 )
228 Ptp1004 *ptp1004; /* Block of parameter */
229
230 Entr /*@ TP_POSITION */
231
232 /*****/
233 /* Declaration */
234 /*****/
235
236 FASTAUTO Rtletp *hpointer; /* Pointer to the list element */
237 FASTAUTO Pthead *tpthead; /* TP-Standard Head */
238
239 /* Useful definition for MA-Makros */
240 AUTO Char log_text[TEST_LOGTEXT]; /* Logtext */
241 STATIC String fname = "tp2_004";
242 AUTO Char params[TPPAR_LEN];
243
244 /*****/
245 /* Initialisation */
246 /*****/
247
248 Begin /*@*/
249
250 tpthead = ptp1004->pthead;
251 tpthead->ret_code = NO_ERR;
252 tpthead->version = TPVERSION;
253
254 /*@ MA_LOG - Call */
255 sprintf(params, "%d %d", *(ptp1004->tpopenid), *(ptp1004->tpoffset));
256 ENTRY ( fname, params );
257
258 /*****/
259 /*@ Check the INPUT parameter */
260 /*****/
261 /* Check the version */
262 If
263 ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tpthead->version))
264 Then
265
266 tpthead->ret_code = CALL_ERR;
267 tpthead->err_class = ILL_VERS;
268 tpthead->err_value = TPNOSUPVS;
269 EXIT ( fname, tpthead->ret_code );
270 return ( tpthead->ret_code );
271 Fi
272
273 /* Search the list element in the list */
274
275 hpointer = (Rtletp *)rt4_614_search(tpanchor, tp4_001_search_topenid,
276 ptp1004->topenid);
277
278 If
279 ( hpointer == (Rtletp *)NULL )
280 Then /* Element not found : ERROR */
281 tpthead->ret_code = CALL_ERR;
282 tpthead->err_class = PAR_ERR;
283 tpthead->err_value = TPOPENID;
284 #ifdef HOWFAR

```



```

288 Fi
289
290 If /* Test the value of the OFFSET */
291 ( ( *(ptp1004->tpoffset) < 0 ) ||
292   ( *(ptp1004->tpoffset) > hpointer->rtleftp.filelength ) )
293 Then
294     tppbhead->ret_code = CALL_ERR;
295     tppbhead->err_class = PAR_ERR;
296     tppbhead->err_value = TPOFFSET;
297 #ifdef HOWFAR
298     EXIT ( fname, tppbhead->ret_code );
299 #endif
300     return ( tppbhead->ret_code );
301 Fi
302
303 /*****
304  /*@      call LSEEK      */
305  *****/
306
307 If /*@ No error in the call of lseek */
308 ( ( lseek(hpointer->rtleftp.dvfileid, *(ptp1004->tpoffset), 0) ) ==
309   *(ptp1004->tpoffset) )
310
311     Then /*@ Update the position of the file pointer*/
312         hpointer->rtleftp.offset = *ptp1004->tpoffset;
313         hpointer->rtleftp.datalength = *ptp1004->tpoffset;
314
315     Else /*@ Ret_code = ERROR*/
316         tppbhead->ret_code = CALL_ERR;
317         tppbhead->err_class = PAR_ERR;
318         tppbhead->err_value = TPOFFSET;
319
320         EXIT ( fname, tppbhead->ret_code );
321
322     return ( tppbhead->ret_code );
323
324 Fi /*@*/
325
326 hpointer->rtleftp.offset = *ptp1004->tpoffset;
327 hpointer->rtleftp.datalength = *ptp1004->tpoffset;
328 /*@ return ( ret_code ) */
329 #ifdef HOWFAR
330 sprintf(log_text, "1.LOG Position of the file pointer %ld File_ID %d",
331 hpointer->rtleftp.tpoffset, hpointer->rtleftp.dvfileid);
332 LOG(fname, log_text);
333 #endif
334 EXIT ( fname, tppbhead->ret_code );
335 return ( tppbhead->ret_code );
336 Bend /*@*/
337 End /*@*/

```

```

1  /*daton *****/
2  /*
3  /* TYPE      : MODULE
4  /*
5  /* NAME      : close.c
6  /*
7  /* AUTOR     : V.Laurent      D AP11
8  /*
9  /* DATE      : 15.12.86
10 /*
11 /* COMPONENTE : TPDU
12 /*
13 /* DOC.-NR.   :
14 /*
15 /*
16 /* PRD#/VERS. :
17 /*
18 /* DESCRIPTION :
19 /*
20 /*
21 /* SYSTEMABHAENGSIGKEIT:
22 /*
23 /*
24 /* HISTORY :
25 /*
26 /* Vers.Nr. | Date | Changes | KZ | CR# | FM# */
27 /* 1.0 | 15.12.86 | Original | V.L |
28 /* | | | | |
29 /*datoff *****/
30
31
32 /*****
33 /*
34 /*          I N C L U D E S
35 /*
36 /*****
37
38
39          /***** Operating System- Includes *****/
40
41 #include <stdio.h>
42 #include <sys/types.h>
43 #include <sys/stat.h>
44
45
46          /***** EM-V1 general Includes *****/
47
48 #include <emsystem.h>
49
50
51
52          /****Part system - extern *****/
53
54
55          /* Interface IMAI */
56 #include <malgcfe.h>
57 #include <magenex.h>
58
59
60
61
62          /***** Teilsystem - intern *****/
63
64          /* TP - Global Definitions */
65 #include <tp2.h>
66
67          /* ENTRY- EXIT and ERROR-Makros */
68 #include <et4100.h>

```



```

72 #include <tp4list.h>
73
74
75
76
77 _EJECT_
78 /*****
79  /*
80  /*          D E F I N E S
81  /*
82  *****/
83
84 #define TPPAR_LEN    90          /* max. length of the loggingtext */
85
86
87
88
89
90 _EJECT_
91 /*****
92  /*
93  /*          D A T A          D E C L A R A T I O N
94  /*
95  *****/
96
97                                     /***** IMPORT *****/
98                                     /* Head of TP-Lists */
99 IMPORT  Rtlh    * tpanchor;
100
101
102                                     /***** EXPORT *****/
103
104 /*NO*/
105
106 _EJECT_
107 /*****
108  /*
109  /*          F U N C T I O N S          D E C L A R A T I O N
110  /*
111  *****/
112
113
114                                     /***** IMPORT *****/
115
116                                     /* Functions of the List */
117 IMPORT  struct Rtle * rt4_614_search ();
118 IMPORT  Void        rt4_615_delete_any ();
119 IMPORT  Short        tp4_001_search_tpopenid ();
120
121
122                                     /* Function of the login*/
123 IMPORT  Short        ma2_003_log ();
124
125
126                                     /***** EXPORT *****/
127
128                                     /* TP-function */
129
130 EXPORT  Short tp2_005_close ();
131
132
133
134
135
136 _EJECT_
137 /*exon *****/
138 /*
139 /* TYPE          : C-FUNKTION
140 /*

```

```

144 /* DATE          : 15.12.86 */
145 /* */ */
146 /* SYNTAX        : Short tp2_OOS_close( pblk ) */
147 /* */ */
148 /* */ */
149 /* DESCRIPTION : */
150 /* */ */
151 /*          Close a file */
152 /* */ */
153 /* INPUT PARAMETER : */
154 /* */ */
155 /*          Pbhead *pbhead          Standard Head */
156 /*          Short *tpopenid        File Identifier */
157 /* */ */
158 /* */ */
159 /* OUTPUT PARAMETER : */
160 /* */ */
161 /*          no */
162 /* */ */
163 /* RETURNVALUE    : */
164 /* */ */
165 /*          ret_code = NO_ERR */
166 /*          = TEMP_ERR */
167 /*          = CALL_ERR */
168 /* */ */
169 /* STANDARD HEAD  : */
170 /* */ */
171 /*          Version = TP2V01 */
172 /* */ */
173 /*          ret_code = TEMP_ERR */
174 /*          err_class = SYS_ERR */
175 /*          err_value = TPCFERR      Error in the call of the c */
176 /*                                     fonction */
177 /*          ret_code = CALL_ERR */
178 /*          err_class = PAR_ERR */
179 /*          err_value = TPOPENID : Unknown openid */
180 /* */ */
181 /* */ */
182 /*          err_class = ILL_VERS */
183 /*          err_value = TPNOSUPVS : not allowed number of version*/
184 /* */ */
185 /* */ */
186 /* GLOBAL DATA (produced): */
187 /* */ */
188 /*          no */
189 /* */ */
190 /* */ */
191 /* GLOBAL DATA (changed): */
192 /* */ */
193 /*          keine */
194 /* */ */
195 /* */ */
196 /* CALLS FUNCTIONS/PROGRAM/MAKROS: */
197 /* */ */
198 /*          close */
199 /* */ */
200 /*          rt4_614_search */
201 /*          rt4_615_delete_any */
202 /* */ */
203 /*          CHECK_VERSION */
204 /*          ENTRY */
205 /*          EXIT */
206 /* */ */
207 /* */ */
208 /* REMARKS : */
209 /*          When the file is closed with succes , the list element */
210 /*          corresponding to this file is retrieved of the list. */
211 /* */ */
212 /* ***** */

```



```

---
216 Short tp2_005_close( ptp1005 )
217 Ptp1005 *ptp1005; /* Block of parameter */
218
219 Entr /*@ TP_CLOSE */
220
221 /*****
222 /* Declaration */
223 /*****
224
225 FASTAUTO Rtletp *hpointer; /* Pointer to the list element */
226 FASTAUTO Pbhead *tpbhead; /* TP-Standard Head */
227
228 /* Useful definition for MA-Makros */
229 STATIC String fname = "tp2_005";
230 AUTO Char params[TPPAR_LEN];
231
232 /*****
233 /* Initialisation */
234 /*****
235
236 Begin /*@*/
237
238 tpbhead = ptp1005->pbhead;
239 tpbhead->ret_code = NO_ERR;
240
241 /*@ MA_LOG - Call */
242 sprintf(params, "%d", *(ptp1005->tpopenid));
243 ENTRY (fname, params);
244
245 /*****
246 /*@ Check the INPUT parameter */
247 /*****
248 /* Check the version */
249
250 IF
251 ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tpbhead->version))
252 Then
253
254 tpbhead->ret_code = CALL_ERR;
255 tpbhead->err_class = ILL_VERS;
256 tpbhead->err_value = TPNOSUPVS;
257 EXIT ( fname, tpbhead->ret_code );
258 return ( tpbhead->ret_code );
259 Fi
260
261 /* Search the list element */
262 hpointer = (Rtletp *)rt4_614_search( tpanchor, tp4_001_search_topenid,
263 ptp1005->tpopenid );
264
265 IF
266 ( hpointer == (Rtletp *)NULL )
267 Then /* Element not found : ERROR */
268 tpbhead->ret_code = CALL_ERR;
269 tpbhead->err_class = PAR_ERR;
270 tpbhead->err_value = TPOPENID;
271 #ifdef HOWFAR
272 EXIT ( fname, tpbhead->ret_code );
273 #endif
274 return ( tpbhead->ret_code );
275 Fi
276 /*****
277 /*@ call CLOSE */
278 /*****
279
280 IF /*@ */
281 ( ( close(hpointer->rtletp.tpfileid) ) == 0 )
282 Then /*@ */
283
284 /*****

```

```

288      rt4_615_delete_any ( tpanchor, (struct Rtle *)hpinter);
289 Else      /*@ Error in the execution of the function CLOSE  */
290
291
292      tppbhead->ret_code = TEMP_ERR;
293      tppbhead->err_class = SYS_ERR;
294      tppbhead->err_value = TPOCFERR;
295  Fi      /*@*/
296          /*@ return (ret_code) */
297  EXIT ( fname, tppbhead->ret_code );
298  return ( tppbhead->ret_code );
299  Bend      /*@*/
300  End      /*@*/

```



```

1 /*daton *****/
2 /*
3 /* TYPE : MODULE
4 /*
5 /* NAME : delete
6 /*
7 /* AUTOR : V.Laurent D AP11
8 /*
9 /* DATE : 11.12.86
10 /*
11 /* COMPONENTE : TPDU
12 /*
13 /* DOC.-NR. :
14 /*
15 /*
16 /* PRD#/VERS. :
17 /*
18 /* DESCRIPTION :
19 /*
20 /*
21 /* SYSTEMARHAENGIGKEIT:
22 /*
23 /*
24 /* HISTORY :
25 /*
26 /* Vers.Nr. | Date | Changes | KZ | CR# | FM# */
27 /* 1.0 | 11.12.86 | Original | V.L. |
28 /* | | | | |
29 /*datoff *****/
30
31
32
33 /*****
34 /*
35 /* INCLUDES
36 /*
37 /*****
38
39 /***** Operating System - Includes *****/
40
41
42 #include <stdio.h>
43 #include <sys/types.h>
44 #include <sys/stat.h>
45 #include <sys/fcntl.h>
46
47 /***** EM-V1 general Includes *****/
48
49 #include <emsystem.h>
50
51
52
53 /*****Part system - extern *****/
54
55
56
57
58
59 /* Interface IMAI */
60 #include <malgcfe.h>
61 #include <magenex.h>
62
63
64
65
66 /*****Part system - intern *****/
67
68 /* TP - Rights Definitions */

```

```

72                                     /* ENTRY- EXIT and ERROR-Makros */
73 #include <rt4log.h>
74
75                                     /* Listen */
76 #include <rt4list.h>
77 #include <tp4list.h>
78
79
80 _EJECT_
81 /*****
82 /*
83 /*          D E F I N E S
84 /*
85 /*****
86
87
88                                     /***** Constants *****/
89
90                                     /* max. Laength of a Text for the Login File */
91 #define TPAR_LEN      90
92 #define RTLN_LOGTEXT  60
93 #define TEST_LOGTEXT 150
94
95                                     /******Error text*****
96
97 #define TP2ERRROD      " Error in f_length"
98
99
100                                     /******MODE OF CREATION*****
101
102 #define COUT 00777
103
104
105
106
107
108 _EJECT_
109 /*****
110 /*
111 /*          DATA          D E C L A R A T I O N
112 /*
113 /*****
114
115                                     /***** IMPORT *****/
116
117                                     /* Head of TP-Liste */
118 IMPORT Rtlh      * tpanchor;
119
120                                     /***** EXPORT *****/
121
122
123 /* NO */
124
125 _EJECT_
126 /*****
127 /*
128 /*          FUNCTIONS          D E C L A R A T I O N
129 /*
130 /*****
131                                     /* IMPORT */
132
133                                     /* Functions for the list */
134 IMPORT struct Rtle * rt4_612_create_and_append ();
135 IMPORT struct Rtle * rt4_614_search ();
136 IMPORT Void      rt4_615_delete_any ();
137 IMPORT Short      tp4_001_search_tpcopenid ();
138
139
140                                     /* Function for the login */

```



```

144
145                                     /***** EXPORT *****/
146
147                                     /* tp-Functions */
148 EXPORT Short tp2_Q06_delete ();
149 EXPORT Short tp4_Q16_delete_all ();
150 EXPORT Short tp4_Q26_delete_part ();
151
152
153
154 _EJECT_
155 /*exon *****/
156 /*
157 /* TYPE          : C-FUNKTION
158 /*
159 /* NAME          : tp2_Q06_delete
160 /*
161 /* AUTOR         : V.Laurent ,          DAP 11
162 /* DATE          : 31.10.86
163 /*
164 /* SYNTAX        : Short tp2_Q06_delete( pblk )
165 /*
166 /*
167 /* DESCRIPTION :
168 /*
169 /*          Delete a file ( make it empty ) or
170 /*          a prt of a file (since a certain position
171 /*          to the end of the file).
172 /*
173 /*
174 /*
175 /* INPUT PARAMETER :
176 /*
177 /*          Pthead *pthead          Standard Head
178 /*          Short  *tpopenid        File Identifier
179 /*          Long   *tprequired_data_length The number of non
180 /*                                          delete bytes since the
181 /*                                          beginning of the file
182 /*
183 /* OUTPUT PARAMETER :
184 /*
185 /*          Long   *tpdata_length    The actual length of the file
186 /*
187 /*
188 /* RETURNVALUE :
189 /*
190 /*          Short  ret_code = NO_ERR
191 /*                      = TEMP_ERR
192 /*                      = CALL_ERR
193 /*
194 /* STANDARD HEAD :
195 /*
196 /*          Version = TP2V01
197 /*
198 /*          ret_code = NO_ERR
199 /*
200 /*          ret_code = TEMP_ERR
201 /*          err_class = SYS_ERR
202 /*          err_value = TPCFERR : Error in the execution
203 /*                               of the c function.
204 /*          ret_code = CALL_ERR
205 /*          err_class = PAR_ERR
206 /*          err_value = TPOPENID : Unknown File Identifier
207 /*          err_value = TPDELREQUIRED : The number of bytes to
208 /*                                     not delete is wrong.
209 /*
210 /*          err_class = ILL_VERS
211 /*          err_value = TPNOSUPVS :Not allowed number of version
212 /*

```

```

216 /*          no                                     */
217 /*          */
218 /* GLOBAL DATA (changed) :                         */
219 /*          */
220 /*          no                                     */
221 /*          */
222 /*          */
223 /* CALLS FUNCTIONS/PROGRAM/MAKROS:                 */
224 /*          */
225 /*          tp4_016_delete_all                      */
226 /*          tp4_026_delete_part                    */
227 /*          */
228 /*          rt4_614_search                          */
229 /*          */
230 /*          CHECK_VERSION                           */
231 /*          ENTRY                                   */
232 /*          EXIT                                    */
233 /*          */
234 /*          */
235 /* REMARKS :                                         */
236 /*          */
237 /*          */
238 /*exoff *****/
239 _EJECT_
240
241
242 Short tp2_006_delete( ptp1006 )
243 Ptp1006 *ptp1006;                                     /* Block of parameters */
244
245 Entr                                           /*@ TP_DELETE */
246
247 /*****/
248 /*          Declaration                            */
249 /*****/
250
251 FASTAUTO Rtletp *hpointer;                       /* Pointer to Listenement */
252 AUTO Char log_text[TEST_LOGTEXT];                /* Logtext */
253 Short rc;                                         /* return code */
254 FASTAUTO Pthead *tppbhead;                       /* TP-Standard Head */
255
256                                     /* Useful definitions for MA-Makros */
257
258 STATIC String fname = "tp2_006";
259 AUTO Char params[TPPAR_LEN];
260
261 /*****/
262 /*          Initialisation                            */
263 /*****/
264
265
266 Begin                                           /*@*/
267
268                                     /*@ MA_LOG - call */
269 sprintf(params, "%d %ld", *ptp1006->tpopenid,
270 *ptp1006->tprequired_data_length);
271 ENTRY ( fname, params );
272 tppbhead = ptp1006->pthead;
273
274
275 /*****/
276 /*@          Check the INPUT parameters                            */
277 /*****/
278
279                                     /* Check the version */
280
281 If
282 ( ! CHECK_VERS (TP2MINVERS, TP2MAXVERS, tppbhead->version))
283 Then
284

```



```

288      EXIT ( fname, tppbhead->ret_code );
289      return ( tppbhead->ret_code );
290  Fi
291
292
293
294                                     /*@ Search the list element */
295  hpointer = (Rtletp *)rt4_614_search
296      ( tpanchor , tp4_001_search_tpopenid , ptp1006->tpopenid );
297
298  If
299      ( hpointer == (Rtletp *)NULL )
300  Then                                     /* Element not found */
301      tppbhead->ret_code = CALL_ERR;
302      tppbhead->err_class = PAR_ERR;
303      tppbhead->err_value = TPOPENID;
304  #ifdef HOWFAR
305      EXIT ( fname, ptp1006->pbhead->ret_code );
306  #endif
307      return ( ptp1006->pbhead->ret_code );
308  Fi
309
310
311
312  If
313      ( hpointer->rtletbp.modus != TPOUT )
314  Then                                     /* The file is opened for reading */
315      tppbhead->ret_code = CALL_ERR;
316      tppbhead->err_class = PAR_ERR;
317      tppbhead->err_value = TPDIRUSE;
318      *ptp1006->tpdata_length = hpointer->rtletbp.filelength;
319  #ifdef HOWFAR
320      EXIT ( fname, tppbhead->ret_code );
321  #endif
322      return ( tppbhead->ret_code );
323  Fi
324
325
326
327  If
328      ( ((*ptp1006->tprequired_data_length) < 0 ) ||
329        ((*ptp1006->tprequired_data_length) > hpointer->rtletbp.filelength))
330  Then
331      /* Number of non delete bytes is < 0 or > file length */
332      ptp1006->pbhead->ret_code = CALL_ERR;
333      ptp1006->pbhead->err_class = PAR_ERR;
334      ptp1006->pbhead->err_value = TPDELREQUIRED;
335      *ptp1006->tpdata_length = hpointer->rtletbp.filelength;
336      EXIT ( fname, ptp1006->pbhead->ret_code );
337      return ( ptp1006->pbhead->ret_code );
338  Fi
339
340  /*****
341  /*      Number of non delete bytes = file length      */
342  *****/
343
344
345  If                                     /*@ Number of non delete bytes = file length*/
346      ( *ptp1006->tprequired_data_length == hpointer->rtletbp.filelength )
347  Then                                     /*@ Ret_code = NO ERROR */
348      *ptp1006->tpdata_length = hpointer->rtletbp.filelength;
349      ptp1006->pbhead->ret_code = NO_ERR;
350      EXIT ( fname, ptp1006->pbhead->ret_code );
351      return ( ptp1006->pbhead->ret_code );
352  Fi                                     /*@*/
353
354  /*****
355  /*      Selection of the delete function      */
356  *****/

```



```

360
361      If                                     /*@ Number of non delete byte = 0*/
362          ( *ptp1006->tprequired_data_length == 01 )
363
364      Then                                     /*@ call tp-delete-all */
365
366          rc = tp4_016_delete_all ( ptp1006 , hpointer );
367
368
369          Switch ( rc )
370
371              Case NO_ERR :
372                  *ptp1006->tpdata_length = *ptp1006->tprequired_data_length;
373                  break;
374              Default :
375                  break;
376          Esac
377
378      Else                                     /*@ Number of non delete byte is < file length */
379          /*@ call tp-delete-part */
380
381
382          rc = tp4_026_delete_part ( ptp1006 , hpointer );
383
384
385          Switch ( rc )
386              Case NO_ERR :
387                  *ptp1006->tpdata_length = *ptp1006->tprequired_data_length;
388                  break;
389              Default :
390                  break;
391          Esac
392      Fi                                     /*@*/
393
394      #ifdef HOWFAR
395      sprintf(log_text,"1.LOG not delete bytes %ld filelength %ld",
396          *ptp1006->tprequired_data_length, *ptp1006->tpdata_length);
397      LOG(fname,log_text);
398      #endif
399
400                                     /*@return ( ret_code ) */
401      EXIT ( fname, ptp1006->pbhead->ret_code );
402      return ( ptp1006->pbhead->ret_code );
403      Bend                                     /*@*/
404      End                                     /*@*/
405
406      _EJECT_
407      /*exon *****/
408      /*
409      /* TYPE          : C-FUNKTION
410      /*
411      /* NAME          : tp4_016_delete_all
412      /*
413      /* AUTHOR        : V.Laurent, D AP 11
414      /* DATE          : 11.12.86
415      /*
416      /* SYNTAX        : Short tp4_016_delete_all ( pblk , hpointer )
417      /*
418      /*
419      /* DESCRIPTION   :
420      /*
421      /*              Delete all the file (make it empty)
422      /*
423      /* INPUT PARAMETER :
424      /*
425      /*      Pbhead *pbhead          Standard Head
426      /*
427      /*
428      /*      Pblk *hpointer          Pointer to the block head

```



```

432 /* */
433 /* */
434 /* RETURNVALUE : */
435 /* */
436 /* Short ret_code = NO_ERR */
437 /* = TEMP_ERR */
438 /* */
439 /* */
440 /* STANDARD HEAD : */
441 /* */
442 /* Version = TP2V01 */
443 /* */
444 /* ret_code = TEMP_ERR */
445 /* err_class = SYS_ERR */
446 /* err_value = TPCEFERR Error in the execution of the */
447 /* c function */
448 /* */
449 /* */
450 /* */
451 /* GLOBAL DATA (produced): */
452 /* */
453 /* no */
454 /* */
455 /* GLOBAL DATA (changed) : */
456 /* */
457 /* no */
458 /* */
459 /* */
460 /* CALLS FUNCTIONS/PROGRAM/MAKROS: */
461 /* */
462 /* creat */
463 /* close */
464 /* unlink */
465 /* */
466 /* ENTRY */
467 /* EXIT */
468 /* */
469 /* */
470 /* REMARKS : */
471 /* To make empty a file , it is closed , erased */
472 /* a new file (empty) is created with the name of the old */
473 /* deleted one. */
474 /* */
475 /* */
476 /* */
477 /*exoff *****/
478 _EJECT_
479
480
481 Short tp4_016_delete_all( ptp1006 , hpointer )
482 Ptp1006 *ptp1006; /* Block of parameter */
483 Rtlebtp *hpointer; /* Pointer to a List element */
484
485 Entr /*& TP_DELETE_ALL */
486
487 /* *****/
488 /* Declaration */
489 /* *****/
490
491 AUTO Char log_text[TEST_LOGTEXT]; /* Logtext */
492 /* Useful parameter for MA-Makros */
493 STATIC String fname = "tp4_016";
494 AUTO Char params[TPPAR_LEN];
495 FASTAUTO Short fdes; /*Return value of CREAT in delete_all*/
496
497 /* *****/
498 /* Initialisation */
499 /* *****/
500

```



```

504                                     /*@ MA_LOG - call */
505 sprintf(params, "%s", "");
506 ENTRY ( fname, params );
507
508
509 /*****
510 /*          Close the file          */
511 /*****
512
513
514 /*****
515 /*          call    CLOSE_          */
516 /*****
517
518 If                                     /*@ Error in the execution of close */
519 ( ( close(hpointer->rtleftp.dvfileid) ) != 0 )
520 Then                                     /*@ Ret_code = TEMP_ERR */
521     ptp1006->pbhead->ret_code = TEMP_ERR;
522     ptp1006->pbhead->err_class = SYS_ERR;
523     ptp1006->pbhead->err_value = TPCEFERR;
524 #ifdef HOWFAR
525     EXIT ( fname, ptp1006->pbhead->ret_code );
526 #endif
527     return ( ptp1006->pbhead->ret_code );
528 Fi                                     /*@*/
529
530 /*****
531 /*          Delete the file          */
532 /*****
533
534 /*****
535 /*          call    UNLINK          */
536 /*****
537
538 If                                     /*@ Error in the removing */
539 ( ( unlink(hpointer->rtleftp.filename) ) != 0 )
540 Then                                     /*@ Ret_code = TEMP_ERR */
541     ptp1006->pbhead->ret_code = TEMP_ERR;
542     ptp1006->pbhead->err_class = SYS_ERR;
543     ptp1006->pbhead->err_value = TPCEFERR;
544 #ifdef HOWFAR
545     EXIT ( fname, ptp1006->pbhead->ret_code );
546 #endif
547     return ( ptp1006->pbhead->ret_code );
548 Fi                                     /*@*/
549
550 /*****
551 /*          Creat a new file with the same name          */
552 /*****
553
554 /*****
555 /*          Call    CREAT          */
556 /*****
557 fdes = creat(hpointer->rtleftp.filename, COU);
558 If                                     /*@ Error in the creation */
559 ( ( fdes ) == -1 )
560 Then                                     /*@ Ret_code = TEMP_ERR */
561     ptp1006->pbhead->ret_code = TEMP_ERR;
562     ptp1006->pbhead->err_class = SYS_ERR;
563     ptp1006->pbhead->err_value = TPCEFERR;
564
565     EXIT ( fname, ptp1006->pbhead->ret_code );
566     return ( ptp1006->pbhead->ret_code );
567 Fi                                     /*@*/
568
569 /*****
570 /*          Update the list element corresponding to this file          */
571 /*****
572 hpointer->rtleftp.dvfileid = fdes;

```



```

576 hpointer->rtleabtp.modus = TPOUT;
577
578 #ifdef HOWFAR
579 sprintf(log_text,"11.LOG not delete %ld filelength %ld",
580 *ptp1006->tprequired_data_length, *ptp1006->tpdata_length);
581 LOG(fname,log_text);
582 #endif
583
584 /*@ return */
585 EXIT ( fname, NO_ERR );
586 return ( NO_ERR );
587 End /*@*/
588 End /*@*/
589
590
591
592
593 _EJECT_
594 /*exon *****/
595 /*
596 /* TYPE : C-FUNKTION
597 /*
598 /* NAME : tp4_026_delete_part
599 /*
600 /* AUTOR : V.Laurent, D AP 11
601 /* DATE : 11.12.86
602 /*
603 /* SYNTAX : Short tp4_026_delete_part ( pblk , hpointer )
604 /*
605 /*
606 /* DESCRIPTION :
607 /*
608 /* Delete a number of bytes between the actual
609 /* position of the file pointer and the end of file.
610 /*
611 /* INPUT PARAMETER :
612 /*
613 /* Pthead *pthead Standard Head
614 /* Long *tprequired_data_length The number of not
615 /* delete bytes since
616 /* the beginning
617 /* of the file.
618 /* Rtlebtp *hpointer Pointer to the
619 /* list element.
620 /*
621 /* OUTPUT PARAMETER :
622 /*
623 /*
624 /* RETURNVALUE :
625 /*
626 /* Short ret_code = NO_ERR
627 /* = TEMP_ERR
628 /*
629 /*
630 /* STANDARD HEAD :
631 /*
632 /* Version = TP2V01
633 /*
634 /* ret_code = TEMP_ERR
635 /* err_class = SYS_ERR
636 /* err_value = TPCEFERR : Error in the execution of
637 /* a c function.
638 /*
639 /*
640 /*
641 /*
642 /*
643 /*
644 /* GLOBAL DATA (continued)

```

```

648 /* GLOBAL DATA (changed) : */
649 /* */
650 /*      no */
651 /* */
652 /* */
653 /* */
654 /* CALLS FUNCTIONS/PROGRAM/MMAKROS: */
655 /* */
656 /*      unlink */
657 /*      open */
658 /*      read */
659 /*      write */
660 /*      close */
661 /*      creat */
662 /*      link */
663 /*      fstat */
664 /*      mktemp */
665 /* */
666 /*      rt4_b12_create_and_append */
667 /*      rt4_b14_search */
668 /*      rt4_b15_delete_any */
669 /* */
670 /*      tp4_Q01_search_tpopenid */
671 /* */
672 /* */
673 /*      ENTRY */
674 /*      EXIT */
675 /*      ERROR */
676 /* */
677 /* */
678 /* REMARKS : */
679 /*      The file is closed and reopened with the reading mode */
680 /*      A workfile is created and opened in writing mode */
681 /*      The number of not delete bytes is copied in the workfile */
682 /*      with packets of 1024 bytes. The old file is closed and */
683 /*      released and the workfile is renamed with the name of the */
684 /*      old one. The new file is closed and reopened in the writing */
685 /*      mode. The length of the file is calculated to verify if */
686 /*      all the bytes are been, with succdes, written. */
687 /* */
688 /* */
689 /* */
690 /*exoff *****/
691 _EJECT_
692
693
694
695
696
697 Short tp4_Q06_delete_part ( ptp1006 , hpointer )
698 Ptp1006 *ptp1006; /* Block of parameter */
699 Rtletp *hpointer; /* Pointer to a list element */
700
701 Entr /*@ OP_DELETE_PART */
702
703 /*****/
704 /*      Declaration */
705 /*****/
706
707 FASTAUTO Long lgthfile; /* The new file length */
708 AUTO Char *mktemp(); /* A pointer to the return of mktemp */
709 FASTAUTO struct stat vtpinf; /*The structure returns by fstat*/
710 FASTAUTO Short fndes; /* New identifier of the new file */
711 FASTAUTO Short fold; /*Identifier of the old file */
712 FASTAUTO int fwdes; /*Idebtifier of the work file */
713 FASTAUTO Short rest; /*Rest of bytes to copy */
714 FASTAUTO Rtletp *hpointer; /* Pointer to a list element */
715 FASTAUTO Long newfilelength; /*New file length*/
716 AUTO Char storage[RUEF17F1].

```



```

720
721                                     /* Useful definitions for MA-Makros */
722 STATIC String fname = "tp4_Q26";
723 AUTO      Char  params[TPPAR_LEN];
724
725 /*****
726 /*      Initialisation
727 /*****
728
729
730 Begin                                     /*@*/
731
732                                     /*@ MA_LOG call      */
733 sprintf(params, "%ld", *(ptp1006->tprequired_data_length));
734 ENTRY ( fname , params );
735
736                                     /* Close the old file for opening with the read mode */
737
738
739 /*****
740 /*@      call      CLOSE
741 /*****
742
743 If                                     /*@ Error in the execution of close */
744     ( ( close(hpointer->rtletp.dvfileid) ) != 0 )
745 Then                                     /*@ Ret_code = TEMP_ERR */
746     ptp1006->pbhead->ret_code = TEMP_ERR;
747     ptp1006->pbhead->err_class = SYS_ERR;
748     ptp1006->pbhead->err_value = TPCEFERR;
749 #ifdef HOWFAR
750     EXIT ( fname, ptp1006->pbhead->ret_code );
751 #endif
752     return ( ptp1006->pbhead->ret_code );
753 Fi                                     /*@*/
754
755 /*****
756 /*@      call      OPEN
757 /*****
758
759
760
761 fold = open(hpointer->rtletp.filename,O_RDONLY);
762 If                                     /*@ Error in the opening of the old file*/
763     ( ( fold ) < 0 )
764 Then                                     /*@ Retrieve the list-element */
765                                     /*@ ret_code = ERROR */
766     rt4_615_delete_any (tpanchor, (struct Rtle *)hpointer);
767
768     ptp1006->pbhead->ret_code = TEMP_ERR;
769     ptp1006->pbhead->err_class = SYS_ERR;
770     ptp1006->pbhead->err_value = TPCEFERR;
771 #ifdef HOWFAR
772     EXIT ( fname, ptp1006->pbhead->ret_code );
773 #endif
774     return ( ptp1006->pbhead->ret_code );
775 Else                                     /*@*/
776
777 /*****
778 /*@      Complete the list_element corresponding to this file */
779 /*****
780
781     hpointer->rtletp.dvfileid = fold;
782     hpointer->rtletp.modus    = TPIN;
783     hpointer->rtletp.offset   = 0L;
784     hpointer->rtletp.datalength = 0L;
785
786                                     /*@ Subtract from it the number of bytes to delete */

```



```

792
793 /*****
794 /*@ Produce a list element for the work file */
795 /*****/
796
797 hlpinter = (Rtletp *)rt4_612_create_and_append
798             ((Rtlh *)tpanchor, sizeof(Rtletp));
799
800 /*****
801 /*@ Produce a unique filename for the work file */
802 /*****/
803
804
805
806     sprintf (unique,"tpXXXXXX");
807
808     If ( mktemp(unique) == NULL )
809         Then     sprintf (log_text, "File %s exists soon", unique);
810                 LOG (fname,log_text);
811                 ptp1006->pbhead->ret_code = TEMP_ERR;
812                 ptp1006->pbhead->err_class = SYS_ERR;
813                 ptp1006->pbhead->err_value = TPCEFERR;
814 #ifdef HOWFAR
815     EXIT ( fname, ptp1006->pbhead->ret_code );
816 #endif
817     return ( ptp1006->pbhead->ret_code );
818
819
820     Fi
821
822
823 hlpinter->rtletp.filename = unique;
824
825
826                                     /* Creat the work file */
827 /*****
828 /*@ Call CREAT */
829 /*****/
830 fwdx = creat(unique ,OOUT);
831 If
832     ( ( fwdx ) < 0 )
833 Then
834                                     /*@ Ret_code = TEMP_ERR */
835     ptp1006->pbhead->ret_code = TEMP_ERR;
836     ptp1006->pbhead->err_class = SYS_ERR;
837     ptp1006->pbhead->err_value = TPCEFERR;
838 #ifdef HOWFAR
839     EXIT ( fname, ptp1006->pbhead->ret_code );
840 #endif
841     return ( ptp1006->pbhead->ret_code );
842
843                                     /*@*/
844 /*****
845 /*@ Complete the list element corresponding to the work file */
846 /*****/
847 hlpinter->rtletp.tpfileid = hlpinter->rtletp.tpfileid;
848 hlpinter->rtletp.dvfileid = fwdx;
849 hlpinter->rtletp.modus = TPOUT;
850 hlpinter->rtletp.offset = 01;
851 hlpinter->rtletp.datalength = 01;
852 hlpinter->rtletp.filelength = 01;
853
854 /*****
855 /* Copy the file */
856 /*****/
857
858
859
860 While

```

```

1002 If ( ( link(hlpinter->rtletp.filename,
1003             hlpinter->rtletp.filename) ) == -1 )
1004 Then

```



```

864 /*****
865 /*@          call    READ          */
866 /*****
867
868
869 If          ( ( read(fold,storage,BUFSIZE) ) != BUFSIZE )
870 Then
871             /* Error in reading the packets*/
872             ptp1006->pbhead->ret_code = TEMP_ERR;
873             ptp1006->pbhead->err_class = SYS_ERR;
874             ptp1006->pbhead->err_value = TPCEFERR;
875 #ifdef HOWFAR
876             EXIT ( fname, ptp1006->pbhead->ret_code );
877 #endif
878             return ( ptp1006->pbhead->ret_code );
879
880             /*@ Update the datalength*/
881             hpointer->rtlenbtp.datalength += 1024;
882
883 /*****
884 /*@          Call    WRITE          */
885 /*****
886
887 If          ( ( write(fwdes,storage,BUFSIZE) ) != BUFSIZE )
888 Then
889             /* Error in writing in the work file*/
890             ptp1006->pbhead->ret_code = TEMP_ERR;
891             ptp1006->pbhead->err_class = SYS_ERR;
892             ptp1006->pbhead->err_value = TPCEFERR;
893 #ifdef HOWFAR
894             EXIT ( fname, ptp1006->pbhead->ret_code );
895 #endif
896             return ( ptp1006->pbhead->ret_code );
897
898             /*@
899
900             /*@ Verify if there is still less than 1024 bytes to copy */
901             rest = (int)(newfilelength - hpointer->rtlenbtp.datalength);
902
903
904
905 If          /*@ There is more than 0 byte*/
906             ( ( rest) > 0 )
907 Then
908             /*@ Copy the rest into the work file */
909
910             /* Read the rest of the file */
911
912
913 /*****
914 /*@          call    READ          */
915 /*****
916 If          ( ( read(fold,storage,rest) ) != rest )
917 Then
918             /* Error in reading the rest of the file */
919             ptp1006->pbhead->ret_code = TEMP_ERR;
920             ptp1006->pbhead->err_class = SYS_ERR;
921             ptp1006->pbhead->err_value = TPCEFERR;
922 #ifdef HOWFAR
923             EXIT ( fname, ptp1006->pbhead->ret_code );
924 #endif
925             return ( ptp1006->pbhead->ret_code );
926
927
928             /* Write the rest into the work file */
929
930
931 /*****
932 /*@          call    WRITE          */

```



```

936         Then                                     /* Error in writing the rest*/
937             ptp1006->pbhead->ret_code = TEMP_ERR;
938             ptp1006->pbhead->err_class = SYS_ERR;
939             ptp1006->pbhead->err_value = TPCEFERR;
940 #ifdef HOWFAR
941             EXIT ( fname, ptp1006->pbhead->ret_code );
942 #endif
943             return ( ptp1006->pbhead->ret_code );
944     Fi
945 Fi
946
947                                     /*3 Close the old file */
948 /*****
949 /*          call    CLOSE
950 *****/
951
952 If      ( ( close(fold) ) != 0 )
953     Then                                     /* Error in the execution of close */
954         ptp1006->pbhead->ret_code = TEMP_ERR;
955         ptp1006->pbhead->err_class = SYS_ERR;
956         ptp1006->pbhead->err_value = TPCEFERR;
957 #ifdef HOWFAR
958         EXIT ( fname, ptp1006->pbhead->ret_code );
959 #endif
960         return ( ptp1006->pbhead->ret_code );
961 Fi
962
963                                     /*3 Delete the old file */
964 /*****
965 /*          call    UNLINK
966 *****/
967
968 If      ( ( unlink(hpointer->rtleftp.filename) ) != 0 )
969     Then                                     /* Error in deleting the old file */
970         ptp1006->pbhead->ret_code = TEMP_ERR;
971         ptp1006->pbhead->err_class = SYS_ERR;
972         ptp1006->pbhead->err_value = TPCEFERR;
973 #ifdef HOWFAR
974         EXIT ( fname, ptp1006->pbhead->ret_code );
975 #endif
976         return ( ptp1006->pbhead->ret_code );
977 Fi
978
979                                     /*3 Close the new file */
980 /*****
981 /*          call    CLOSE
982 *****/
983
984
985
986 If      ( ( close(fwdes) ) != 0 )
987     Then                                     /* Error in close of the new file*/
988         ptp1006->pbhead->ret_code = TEMP_ERR;
989         ptp1006->pbhead->err_class = SYS_ERR;
990         ptp1006->pbhead->err_value = TPCEFERR;
991 #ifdef HOWFAR
992         EXIT ( fname, ptp1006->pbhead->ret_code );
993 #endif
994         return ( ptp1006->pbhead->ret_code );
995 Fi
996
997                                     /*3 Rename the work file */
998 /*****
999 /*          call    LINK
1000 *****/
1001
1002 If      ( ( link(hipointer->rtleftp.filename,
1003                 hpointer->rtleftp.filename) ) == -1 )
1004     Then                                     /* Error in renaming the work file */

```



```

1008 #ifdef HOWFAR
1009     EXIT ( fname, ptp1006->pbhead->ret_code );
1010 #endif
1011     return ( ptp1006->pbhead->ret_code );
1012 Fi
1013                                     /*@ Remove the work file */
1014
1015 /*****
1016  /*      call UNLINK
1017  *****/
1018
1019 If      ( ( unlink( unique ) ) != 0 )
1020     Then                                     /* Error in deleting the old file */
1021         ptp1006->pbhead->ret_code = TEMP_ERR;
1022         ptp1006->pbhead->err_class = SYS_ERR;
1023         ptp1006->pbhead->err_value = TPCEFERR;
1024 #ifdef HOWFAR
1025     EXIT ( fname, ptp1006->pbhead->ret_code );
1026 #endif
1027     return ( ptp1006->pbhead->ret_code );
1028 Fi
1029
1030                                     /*@ Open again the new file */
1031 /*****
1032  /*      Call OPEN_
1033  *****/
1034
1035     fndes = open( hpointer->rtleftp.filename, O_WRONLY );
1036
1037
1038 If      ( ( fndes ) < 0 )
1039     Then                                     /* Error in the execution of open */
1040         ptp1006->pbhead->ret_code = TEMP_ERR;
1041         ptp1006->pbhead->err_class = SYS_ERR;
1042         ptp1006->pbhead->err_value = TPCEFERR;
1043
1044 #ifdef HOWFAR
1045     EXIT ( fname, ptp1006->pbhead->ret_code );
1046 #endif
1047     return ( ptp1006->pbhead->ret_code );
1048 Fi
1049
1050     hpointer->rtleftp.dvfileid = fndes;
1051
1052                                     /*@ Determine the length of the new file */
1053 /*****
1054  /*      Call FSTAT
1055  *****/
1056
1057
1058 If      ( ( fstat( hpointer->rtleftp.dvfileid, &vtpinf ) ) == -1 )
1059     Then                                     /*@ Error in the execution of fstat*/
1060         ptp1006->pbhead->ret_code = TEMP_ERR;
1061         ptp1006->pbhead->err_class = SYS_ERR;
1062         ptp1006->pbhead->err_value = TPCEFERR;
1063
1064 Else                                     /*@ Verify if the length is > 0*/
1065     lgthfile = (Long)(vtpinf.st_size);
1066     If      ( ( lgthfile ) < 0 )
1067         Then                                     /*@ The length is < 0 */
1068             ptp1006->pbhead->ret_code = TEMP_ERR;
1069             ptp1006->pbhead->err_class = SYS_ERR;
1070             ptp1006->pbhead->err_value = TPNFLERR;
1071             close( hpointer->rtleftp.tpfileid );
1072             class_value.err_class = SYS_ERR;
1073             class_value.err_value = TEMP_ERR;
1074             ERROR ( fname, TPCEFERR, class_value );
1075

```

```

1080          Fi          /*@ Delete the old list element */
1081      Fi          /*@*/
1082
1083          /*@ Complete the list element corresponding to the new file*/
1084      hpointer->rtlabtp.filename = hpointer->rtlabtp.filename;
1085      hpointer->rtlabtp.tpfileid = hpointer->rtlabtp.tpfileid;
1086      hpointer->rtlabtp.modus = TPOUT;
1087      hpointer->rtlabtp.offset = 01;
1088      hpointer->rtlabtp.datalength = hpointer->rtlabtp.filelength;
1089
1090
1091          /*@ Delete the old list element */
1092      rt4_615_delete_any(tpanchor, (struct Rtle *)hpointer);
1093          /*@ Update the variables of the file */
1094          /*@ return ( ret_code ) */
1095      #ifdef HOWFAR
1096      sprintf(log_text, "13.LOG not delete %ld filelength %ld",
1097      *ptp1006->tprequired_data_length, *ptp1006->tpdata_length);
1098      LOG(fname, log_text);
1099      #endif
1100
1101      EXIT ( fname, NO_ERR );
1102      return ( NO_ERR );
1103      Bend          /*@*/
1104      End          /*@*/

```